CrossTalk

December 2006 **The Journal of Defense Software Engineering** Vol. 19 No. 12

Requirements Engineering

Report Documentation Page

Form Approved
OMB No. 0704-0188

1. REPORT DATE **DEC 2006**	2. REPORT TYPE	3. DATES COVERED **00-00-2006 to 00-00-2006**
4. TITLE AND SUBTITLE **CrossTalk: The Journal of Defense Software Engineering. Volume 19, Number 12, December 2006**		5a. CONTRACT NUMBER 5b. GRANT NUMBER 5c. PROGRAM ELEMENT NUMBER
6. AUTHOR(S)		5d. PROJECT NUMBER 5e. TASK NUMBER 5f. WORK UNIT NUMBER
7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES) **OO-ALC/MASE,6022 Fir Ave,Hill AFB,UT,84056-5820**		8. PERFORMING ORGANIZATION REPORT NUMBER
9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)		10. SPONSOR/MONITOR'S ACRONYM(S) 11. SPONSOR/MONITOR'S REPORT NUMBER(S)

12. DISTRIBUTION/AVAILABILITY STATEMENT
Approved for public release; distribution unlimited

13. SUPPLEMENTARY NOTES

14. ABSTRACT

15. SUBJECT TERMS

16. SECURITY CLASSIFICATION OF:			17. LIMITATION OF ABSTRACT	18. NUMBER OF PAGES	19a. NAME OF RESPONSIBLE PERSON
a. REPORT **unclassified**	b. ABSTRACT **unclassified**	c. THIS PAGE **unclassified**	**Same as Report (SAR)**	**32**	

Standard Form 298 (Rev. 8-98)
Prescribed by ANSI Std Z39-18

Requirements Engineering

4 **Twelve Requirements Basics for Project Success**
Based on industry experience; guidance from requirements-related books, articles, and Web sites; and the author's involvement with projects, the author provides a set of 12 requirements basics whose applications will contribute to your project's success.
by Dr. Ralph R. Young

9 **Interpreting Requirements in a He Said/She Said World**
The author shows that when all project stakeholders work from the perspective that this is a collaborative effort then everyone wins.
by Deb Jacobs

14 **Experiences in Eliciting Security Requirements**
This article describes an approach for doing trade-off analysis among requirements elicitation methods.
by Dr. Nancy R. Mead

20 **Requirements as Enablers for Software Assurance**
This article discusses one approach to represent, model, and analyze certification and accreditation requirements to promote a common understanding among stakeholders for engineering more reliable software.
by Dr. Seok-Won Lee and Robin A. Gandhi

Software Engineering Technology

25 **Finding and Fixing Problems Early: A Perspective-Based Approach to Requirements and Design Inspections**
This article describes the application of perspective-based reading, a previously successful early life-cycle software inspection approach to the problem of software security.
by Dr. Jeffrey Carver, Dr. Forrest Shull. and Dr. Ioana Rus

ON THE COVER
Cover Design by Kent Bingham

Additional art services provided by Janna Jensen

Departments

3 **From the Sponsor**

13 **Coming Events**
Letter to the Editor

19 **SSTC 2007 Ad**

29 **2006 Article Index**

31 **BACKTALK**

The CrossTalk staff would like to wish you and yours the very best this holiday season and the happiest of New Years.

CROSSTALK

76 SMXG Co-Sponsor	*Kevin Stamey*
309 SMXG Co-Sponsor	*Randy Hill*
402 SMXG Co-Sponsor	*Diane Suchan*
DHS Co-Sponsor	*Joe Jarzombek*
NAVAIR Co-Sponsor	*Jeff Schwalb*
Publisher	*Brent Baxter*
Associate Publisher	*Elizabeth Starrett*
Managing Editor	*Kase Johnstun*
Associate Editor	*Chelene Fortier-Lozancich*
Article Coordinator	*Nicole Kentta*
Phone	(801) 775-5555
E-mail	crosstalk.staff@hill.af.mil
CrossTalk Online	www.stsc.hill.af.mil/crosstalk

CROSSTALK, The Journal of Defense Software Engineering is co-sponsored by the U.S. Air Force (USAF), the U.S. Department of Homeland Security (DHS), and the U.S. Navy (USN). USAF co-sponsors: Oklahoma City-Air Logistics Center (ALC) 76 Software Maintenance Group (SMXG), Ogden-ALC 309 SMXG, and Warner Robins-ALC 402 SMXG. DHS co-sponsor: National Cyber Security Division of the Office of Infrastructure Protection. USN co-sponsor: Naval Air Systems Command.

The USAF Software Technology Support Center (STSC) is the publisher of CROSSTALK, providing both editorial oversight and technical review of the journal. CROSSTALK's mission is to encourage the engineering development of software to improve the reliability, sustainability, and responsiveness of our warfighting capability.

S T S C

Subscriptions: Send correspondence concerning subscriptions and changes of address to the following address. You may e-mail us or use the form on p. 17.

517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

Article Submissions: We welcome articles of interest to the defense software community. Articles must be approved by the CROSSTALK editorial board prior to publication. Please follow the Author Guidelines, available at <www.stsc.hill.af.mil/crosstalk/xtlkguid.pdf>. CROSSTALK does not pay for submissions. Articles published in CROSSTALK remain the property of the authors and may be submitted to other publications.

Reprints: Permission to reprint or post articles must be requested from the author or the copyright holder and coordinated with CROSSTALK.

Trademarks and Endorsements: This Department of Defense (DoD) journal is an authorized publication for members of the DoD. Contents of CROSSTALK are not necessarily the official views of, or endorsed by, the U.S. government, the DoD, or the STSC. All product names referenced in this issue are trademarks of their companies.

Coming Events: Please submit conferences, seminars, symposiums, etc. that are of interest to our readers at least 90 days before registration. Mail or e-mail announcements to us.

CrossTalk Online Services: See <www.stsc.hill.af.mil/crosstalk>, call (801) 777-0857 or e-mail <stsc.webmaster@hill.af.mil>.

Back Issues Available: Please phone or e-mail us to see if back issues are available free of charge.

Requirements Management Is Required

In 1994, my organization was striving for Capability Maturity Model® (CMM®) Levels 2 and 3. Driving this activity, the Air Force required us to be Level 3 by 1998 or risk losing the ability to compete for workload. While working toward this goal, one of our major challenges was requirements management. Not yet in charge of the 309th, but a middle manager, I attended a group-level meeting where we discussed our struggles with requirements. At that meeting, one of our technical program managers stated that our organization needed to commit to follow a written policy of managing requirements. He discussed the reasons our projects experienced requirement changes, such as nebulous requests from the user, inadequate designs, and in-progress customer requests. The effects of these requirement changes were significant schedule overruns and dissatisfied customers. He mentioned the traps we got ourselves into like accepting work with inadequate guidance and being strong-armed by some customers to deliver very minor or unclear modifications, and taking on high-risk work without proper negotiations or replanning. When this happened, the end result was the customer and our organization both felt cheated because these requirement changes were not documented and plans were not adjusted appropriately.

One of the ironies of this situation is that we actually did have processes for proper requirements management in place, but we simply did not follow them! We had to make significant cultural changes to adequately implement those practices. Remember, the driving forces behind this issue were good ones; we wanted to support our customers and provide all that we could for the warfighter using our products. We had to convince ourselves that in attempting to give everything to everyone, we were actually hurting both the end-user and ourselves. In addition, because of our previous *do anything* approach, we had to re-educate our customers on the need to better manage requirements effectively; we had to be upfront with them on how requirements changes could effect the cost, schedule, and quality of our software products. Perhaps you can see some of your own challenges in our past struggles. The good news is that we did implement the recommendations of that technical program manager, long ago. Now, as a CMM Integration (CMMI®) Level 5 organization, we use our requirements management practices to ensure any requirements changes are approved by both our management and the customer and that our project plans reflect these changes. The end result is that our customers are intimately involved in our software projects. While requirements issues will always remain challenging, both we and our customers are now fully aware of these impacts; the result – we not only have very low schedule variances, but we have satisfied customers.

The articles in this issue of CrossTalk aim to help the readers with their own requirements challenges. In our first article, *Twelve Requirements Basics for Project Success*, Dr. Ralph Young shares insights he has gathered from his own experience as well as reading about the experience of others. Deb Jacobs follows these basics with specific advice on understanding your requirements in *Interpreting Requirements in a He Said/She Said World*. Next, Dr. Nancy Mead gets even more specific as she discusses different requirements elicitation methods in *Experiences in Eliciting Security Requirements*. Our final theme article, *Requirements as Enablers for Software Assurance*, discusses work that Dr. Seok-Won Lee and Robin A. Gandhi have done to consolidate software security requirements from several guidance documents in the Department of Defense in order to identify the applicable set of security requirements necessary for certification and accreditation. Our supporting article by Dr. Jeffrey Carver, Dr. Forrest Shull, and Dr. Ioana Rus suggests performing requirements and design inspections from varying perspectives in *Finding and Fixing Problems Early: A Perspective-Based Approach to Requirements and Design Inspections*.

While all of the practices of the CMMI, AS9100, and ISO 9000 play a role in our current success with software delivery, requirements management was one of the earliest practices that showed visible results. Good requirements engineering requires more than managing existing requirements; it requires effectively eliciting the requirements, verifying requirements are accurate and useful, managing those requirements, and testing the end-product against the requirements. I anticipate the articles in this issue will provide useful insights for both the novice and expert alike.

Randy B. Hill

Randy B. Hill
Co-Sponsor, Ogden Air Logistics Center

Twelve Requirements Basics for Project Success

Dr. Ralph R. Young
Northrop Grumman Information Technology Defense Group

The author provides a set of 12 requirements basics; these recommended approaches will contribute to your project's success. The requirements basics are based on industry experience; guidance from requirements-related books, articles, and Web sites; and the author's involvement with projects. Having an experienced requirements subject matter expert on the project staff can help the project manager and the project team guide investments that will help.

Much has been written about requirements, and surely we have experienced enough to have learned how to do things in ways that support successful project outcomes. Yet, many projects encounter requirements-related problems, most of which could have been avoided. Ample guidance is available to enable successful project management and to perform effective requirements practices [1, 2, 3, 4, 5] that are easy to apply.

Table 1: *Twelve Requirements Basics for Project Success*

1.	Make sure the project staff includes a trained and experienced requirements manager or requirements analyst.
2.	Proactively *partner* with your customer.
3.	Invest in the project's requirements process.
4.	Write a project vision and scope document.
5.	Use proven requirements elicitation techniques such as requirements workshops and prototyping to evolve the real requirements and to gain buy-in from the stakeholders.
6.	Utilize an evolutionary or incremental approach to development, deployment, and implementation of the capabilities.
7.	Use an effective mechanism to control requirements changes and new requirements.
8.	Use an effective automated requirements tool to maintain information about the requirements.
9.	Ensure that the facts concerning the requirements are accurate and that important requirements are not omitted. Ascertain the rationale for every requirement (why it is needed).
10.	Conduct inspections of all requirements-related documents. (*Inspections* are a more rigorous form of peer review.)
11.	Enlist the support and assistance of all members of the project staff in helping to perform *requirements work*.
12.	Proactively address requirements-related risks.

Table 2: *Requirements Secrets*

1.	*Half* of the features provided in most delivered software are never used once [6].
2.	More than half of the effort on most systems and software projects is wasted [6].
3.	The *stated requirements* (i.e., the list of requirements provided to the developer by the customer and users) are *never* the *real requirements*.
4.	Eighty percent of requirements errors found in system testing are a result of incorrect *facts* about the requirements or omitted requirements [7].
5.	Spending a lot of effort on testing may be misguided; it is better to invest more in an effective requirements process that results in higher quality products being provided to test.
6.	Given that a) systems and software development is complex, and b) people (with all our frailties) are involved, the probability of the development effort becoming derailed is very high – unless a proactive effort is made to *partner for success*. See Requirements Basic 2 (page 5) in this article for a description of what is envisioned.
7.	The documents we produce are fraught with errors; the earlier in the development effort those errors (defects) are discovered, the less costly it is to fix them. A peer review process should be implemented to reduce errors. All requirements-related documents should be inspected.
8.	Project managers should pay added attention to the requirements basics and allot sufficient funding for requirements-related activities.
9.	It is neither difficult nor expensive to incorporate inspections in your project's approach [8]; although this technique has been available for a long time, most projects do not use it.
10.	A *defect prevention process* is also easy to train, deploy, and implement. It can save a lot of money and time, but again, most projects do not use it.
11.	Communication on most projects is challenged; a proactive effort to foster good communications is required [1].
12.	An effective automated requirements tool is required to support requirements development and requirements management on all but the smallest project.

Table 1 provides a set of 12 requirements basics for project success with supporting information, suggestions, and recommendations provided in this article. I encourage you to consider them thoughtfully in the context of your project and to determine if some changes in its approach may be worth considering. (Of course, if you digest this article and then decide to make no changes, no improvement opportunities will result. Failure to work proactively to continuously improve is a root cause of a lack of improvement in project success rates [6].) You may relate to the Requirements Secrets provided in Table 2. The sad truth is that these are not really secrets; they are fairly well-known facts that are well documented in the requirements literature (books and articles written by academicians and practitioners concerning requirements). *So, the bottom line is not that we do not know what to do, it is that we do not do it* – a sad commentary on our willingness and commitment to discipline our efforts on projects.

Related to the importance of requirements basics is the need to be agile. How does one balance increased investment in the requirements process with the need to be agile? In my judgment, it is the project's *approach* that makes all the difference. By using an evolutionary or incremental approach, and by delivering versions, releases, and new products, we can be agile. It is not required that we learn a whole new way of doing things.

Twelve Requirements Basics for Project Success

Concerning potential improvements in your project's approach, we need to remind ourselves that each of us is a change agent on our jobs and on projects. Select an improvement opportunity from the following where you have influence and show others that you can be successful.

Basic 1: Provide training for all project participants concerning the requirements processes to be used on the pro-

	Requirements Analyst's Skills Matrix	Junior-Level Analyst	Mid-Level Analyst	Senior-Level Analyst
Skill No.		Knowledge of = K Experience with = X		
1.	Types of requirements.	K	X	X
2.	Criteria for a good requirement.	K	X	X
3.	Customer/user involvement with requirements (joint team).	K	X	X
4.	Identifying real requirements (from the stated requirements).	K	X	X
5.	Anticipating and controlling requirements changes.	K	X	X
6.	Requirements elicitation.	K	X	X
7.	References concerning requirements (books, articles, standards).	K	X	X
8.	Requirements attributes.	K	X	X
9.	Requirements baseline.	K	X	X
10.	Training in systems engineering (e.g., life cycles, risk managment).	K	X	X
11.	Rationale for requirements.	K	X	X
12.	Requirements management tools.	K	X	X
13.	Requirements peer review/inspection.	K	X	X
14.	Requirements syntax.	K	X	X
15.	Requirements traceability.	K	X	X
16.	Requirements verification and validation.	K	X	X
17.	Requirements Review Board/Configuration Review Board/ Configuration Control Board.	K	X	X
18.	Developing and using metrics for requirements activities/processes.	K	X	X
19.	Requirements prioritization.	K	X	X
20.	Technical writing of requirements deliverables (Requirements Traceability Matrix, Software Requirements Specification, Interface Requirements Specification).	K	X	X
21.	Develop, implement, and use requirements processes.		K	X
22.	Quality assurance of requirements.		K	X
23.	Requirements allocation (to components, applications, packages).		K	X
24.	Requirements change control and change notification.		K	X
25.	Requirements repository.		K	X
26.	Requirements errors (missing, incorrect, infeasible, out-of-scope).		K	X
27.	Use-case development (with customer/user).		K	X
28.	Requirements specifications.		X	X
29.	Evaluating requirements for risks.			X
30.	Training the requirements processes.			X
31.	Requirements Impact Estimation Table.			X

Table 3: *RAs Skills Matrix*

ject. Also, there needs to be a technical leader who is highly skilled in requirements engineering. The training and experience required for three levels of a requirements analyst (RA) are provided in Table 3. See [5] for more information and insight, including an extensive set of references to excellent requirements books and articles by many good writers. Industry systems engineering and requirements trainer Robert Halligan believes that the number one problem in requirements engineering is that project managers (PMs) fail to require experienced RAs, and that RAs are not sufficiently trained and/or experienced to perform their roles effectively[1]. PMs can use this skills matrix to recruit and train RAs; RAs can use it to pursue an ongoing professional development program to acquire needed expertise.

It is important to distinguish between a *process* and the *skills* of project professionals. A process may be defined as a set of activities that results in the accomplishment of a task or the achievement of an outcome. Every project uses a requirements process whether it is defined and documented (written down) or not. Some of the advantages of involving a project's key leads in defining and documenting the requirements process for a project include the following:

- Because they helped create it, the people involved in defining and documenting the process acquire a better understanding of it and become more committed to its successful use.
- The leads for other areas within the project become more involved in the requirements activities and bring their expertise and experience to refine the requirements process.
- Once the process is documented, the opportunity exits to improve it based on actual project events.
- The process is likely to be more complete and useable.
- The process is more likely to be integrated into other activities and plans on the project. In other words, by using a collaborative approach, project communication is enhanced.
- Technical performers on projects become more inclined and willing to use an agreed-upon and understood requirements approach.

When addressing the *skills* of a requirements analyst in Table 3, think of these skills *supporting* the project's requirements process. The process goals are created to perform requirements work effectively and successfully; the skills applied by the requirements analyst facilitate the performance of the requirements work. For example, by ensuring that each requirement meets the criteria for a good requirement, the RA enables the project to avoid a lot of rework, thus making the requirements process much more effective. Another example is that by utilizing the *joint team* mechanism (a way to do something), responsibility for the requirements is fixed – all requirements go through a funnel so that accountability for them can be maintained. The joint team mechanism needs to include a few representatives (between two and 12 or more, depending on the size of the project) of both the customer/user and the developer who are empowered to make decisions and take responsibility and accountability for the requirements throughout the project's life cycle.

Basic 2: Proactively partner with your customer. In our work, project practitioners like to use the word *partner*. It suggests that we are collaboratively working toward joint objectives. I am talking about *a unique type of partnering* in which an independent outside facilitator is engaged to orchestrate an approach in which a set of carefully selected stakeholders gain commitment to success through a series of planned partnering workshops[2]. The advantage of this approach is the evolution of a team that is committed to project success – it will not allow the project to become derailed in spite of the inevitable problems and interpersonal issues encountered during the days, weeks, months, and sometimes years of hard (and often conflicted and frustrating) work. Wiegers provides related ideas concerning having a product champion as a specific partnering technique in [9].

Basic 3: Understand the resources required to perform requirements processes effectively and invest in the project's requirements process. The industry average for project investment in its requirements process is 3 percent of project costs; data from NASA shows that when 8-14 percent of project costs are invested in the system life-cycle requirements process, there is a much higher probability of achieving lower costs and improved schedule [10]. The requirements process used by the project or organization should be 1) developed by the project's stakeholders and staff; 2) documented; and 3) continuously improved based on experience on each project. Measure the requirements change activity (requirements volatility), including both new requirements and changes to requirements, to provide insight into project performance. Also, plan for change; this means estimating the amount of change and when it might occur.

Basic 4: Write a project vision and scope document. The benefit of this recommendation is to document the vision of the stakeholders concerning the goals of the project, and to achieve significant consensus on the project's scope (what is included in the project and what is excluded – for example, *product will not support users in Ireland until Vers. 3*). A template for this document is developed by Karl Wiegers [9] and is represented in [15]. In our work, we need to be aware of the need to involve all stakeholders and to gain *buy-in* (commitment to the success of the project). Allowing various stakeholder groups to review the vision and scope document, comment on it, and then revise it to address their stakeholder comments is one technique that can help gain buy-in. One project best practice is to engage stakeholders throughout the project – projects that do so are more successful than those that do not because communication is improved and expectations are more realistic [7]. Also, this provides more opportunities to build interpersonal relationships and even allows customers and users to help solve the problems that arise.

Basic 5: Use proven requirements elicitation techniques such as requirements workshops and prototypes and other forms of visual presentations to evolve the real requirements[3] and to gain stakeholder buy-in [11]. Among more than 40 requirements-gathering techniques, these two seem to be the most effective. In the case of the former, various stakeholder groups will learn more about the perspective of the other stakeholder groups and will have a better understanding of the overall needs to be addressed. Another use of requirements workshops is to review issues and make decisions that best serve all stakeholders. The latter technique is a cost-effective way to gain a better understanding of the customer's and users' real needs – prototypes may be designed, developed, implemented, and updated for a fraction of the cost of a delivered capability. Getting feedback on a visual representation is faster and easier than getting it from text. *The important thing is to evolve the real requirements before starting other technical work*. Experience has shown that using the stated requirements (the requirements provided by the customer and users at the beginning of a development effort) results in an estimated 45 percent rework [12].

Basic 6: Utilize an evolutionary or incremental project approach to development, deployment, and implementation of the needed capabilities. This is both an opportunity and a challenge – whenever one employs a new technique on a real project, additional risk is assumed. My suggestion is to ensure that there are people on the project staff who have previously utilized a new practice, technique, method, tool, or mechanism. In this way, your project can benefit from lessons learned previously (provided, of course, that you pay attention to and are disciplined to incorporate them). An evolutionary or incremental approach allows the project to build hunks of delivered code at a time, and then uses them to learn how and where to proceed. Use versions, deliveries, releases, and new products to accommodate the inevitable requirements changes.

Basic 7: Use an effective mechanism to control requirements changes and new requirements. Controlling changes may mean generating new releases. Most projects utilize Change Control Boards (CCBs), but they usually address detailed project activities such as changes in the code. Use of a higher-level CCB (for example, the joint team I recommend to be responsible for the requirements) to maintain control of the requirements is a good example of applying added discipline on a project. It is logical (though most often not done) that requirements should be prioritized, and that the highest priority and most difficult requirements be addressed first. One reason to do this is that some requirements are *unknowable* at the beginning of a development effort; it requires some development work and related research effort to discover them. It is critical to understand that changes to requirements and new requirements can cause a project to go out of control, thus creating the need for a mechanism to control them. I recommend limiting changes to a maximum of .5 percent per month (6 percent per year) in the capability currently being developed in order to help keep the project under control [13][4].

Basic 8: Use an effective automated requirements tool to maintain information about the requirements. Although many projects do not follow this practice, in my judgment, an automated requirements tool is required for any project except tiny ones. This is because it is necessary to have a lot of information concerning each requirement – its *attributes*. For example, we need to know the following:

- The source of the requirement (who nominated it).
- A unique identifying number for it.
- Its priority (on a scale of one to three).

Table 4: *The Criteria of a Good Requirement*

Each Individual Requirement Should Be:	
Necessary	If the system can meet prioritized real needs without the requirement, it is not necessary.
Feasible	The requirement is *doable* and can be accomplished within available cost and schedule.
Correct	The facts related to the requirement are accurate and it is technically and legally possible.
Concise	The requirement is stated simply.
Unambiguous	The requirement can be interpreted in only one way.
Complete	All conditions under which the requirement applies are stated, and the requirement expresses a whole idea or statement.
Consistent	The requirement is not in conflict with other requirements.
Verifiable	Implementation of the requirement in the system can be proved.
Traceable	The requirement can be traced to its source, and it can be tracked throughout the system (e.g., to the design, code, test, and documentation).
Allocated	The requirement is assigned to a component of the designed system.
Design independent	The requirement does not pose a specific implementation solution.
Non-redundant	The requirement is not a duplicate requirement.
Stated using a standard construct	The requirement is stated as an imperative using the word shall.
Associated with a unique identifier	Each requirement has a unique identifying number.
Devoid of escape clauses	Requirements do not use *if*, *when*, *but*, *except*, *unless*, and *although* and do not include speculative or general terms such as *usually*, *generally*, *often*, *normally*, and *typically*.

- Its relative cost to implement (low, medium, high).
- Its relative difficulty to implement (low, medium, high).
- Each requirement meets the criteria of a good requirement (the criteria shown in Table 4 should be met for each requirement. If they are not, it is likely that the requirement statement is not really correct).
- The *rationale* for the requirement (why is the requirement needed?) [15][5].
- Change history (how has the statement of the requirement changed over the system life?).
- Traceability[6] (of each requirement to its source, as well as the design, code, test, documentation, and training materials).
- Status (draft, final, approved, pending approval, disapproved).
- Assigned to (component of the system).

Two related needs should be considered. The first is that the requirements are used by different people (including customers, users, and project team members) with different *viewpoints* [16]. We need to be able to organize the requirements to provide different perspectives, for example, for customers, users, or testers. The order and the actual requirements selected for these activities are dependent on the viewpoint. The second need is that at any given time, one may need to see all requirements or only those changed ones; an automated requirements tool provides the capability to obtain various reports that are required to perform good requirements work.

Basic 9: Avoid requirements errors. A requirements error is a defect that is discovered in delivered code that is a result of a requirement statement. Data from NASA provided by requirements consultant and trainer Ivy Hooks show that 80 percent of the requirements errors that remain in delivered software are a result of incorrect facts (49 percent) and omitted requirements (31 percent) [10]. This is truly amazing when you think about it – clearly a key opportunity to improve requirements work. Making a concerted effort (investing more in the requirements process) to avoid these two types of requirements errors can save money and time and also improve the quality of delivered capabilities.

Some of the things that can be done to address incorrect facts include the following:

- Provide stakeholder reviews of requirements work products.
- Require that an authoritative source document be specified.
- Require verification of each requirement as part of the planning for testing.

Some of the things that can be done to address omitted requirements include the following:

- Elicit requirements from a variety of stakeholders.
- Conduct requirements workshops and review the requirements collaboratively.
- Conduct formal requirements reviews.
- Ensure that all high-level requirements are addressed and met in the requirements work products.

Basic 10: Utilize inspections of requirements-related documents. Inspections are not difficult to perform, and their use is cost-effective; a concise explanation of how to perform both peer reviews and inspections and how to install a peer review process on a project can be found in [8].

Basic 11: Enlist the support and assistance of all members of the project staff in performing requirements work. It is not just the requirements manager and/or the RA who need to be involved in requirements-related work on a project. Most members of the project staff can help; however, they need to be made aware of *how* they can help and that the PM's expectation and request is that they *do* help. A technique I have used successfully to accomplish this is an *early project requirements briefing* that is made

	Risk	Approach	Suggested Risk Response Strategy
1.	Changing requirements	Mitigation	Implement a high-level CCB (Joint Team); conduct formal requirements reviews such as a System Requirements Review.
2.	Incomplete or missing requirements	Mitigation	Elicit requirements from a variety of stakeholders; conduct requirements workshops; conduct formal requirements reviews; and ensure that all high-level requirements are addressed and met in the requirements work products.
3.	Unclear requirements	Mitigation	Perform requirements analysis and rework existing requirements; provide stakeholder reviews of requirements work products; require that an authoritative source be specified; and require verification of each requirement as part of the planning for the test program.
4.	Invalid requirements (requirements that may not specify what the customer really wants)	Mitigation	Revisit customer; re-establish needs; redevelop requirements; and delete requirements that do not have a good rationale or do not meet the criteria of a good requirement.
5.	Infeasible requirements (requirements technically difficult to implement)	Mitigation	Revisit customer, re-establish needs, redevelop requirements; and delete requirements that do not have a good rationale or do not meet the criteria of a good requirement.
6.	State-of-the-art requirements (something never done before, or your company has never done)	Mitigation	Obtain input from similar projects/companies; perform detailed requirements analysis and find someone who has done it before and obtain advice concerning lessons that were learned from doing something new or in a new way.
7.	Inadequate interface definition	Mitigation	Establish interface management methods (Interface Requirements Documents/Interface Control Documents) and implement responsibility assignment matrices.
8.	Non-verifiable requirements	Mitigation	Involve testers in early requirements development activities; and reword requirements to establish feasible compliance methods.
9.	Incorrectly allocated requirements	Mitigation	Conduct formal requirements reviews and implement a formal requirements management tool.
10.	Non-traceable requirements	Mitigation	Understand traceability more thoroughly and perform detailed requirements analysis to determine potential sources or to eliminate non-traceable requirements.

Table 5: *Suggested Requirements-Related Risk Response Strategies*

to the entire project staff. Participants in the briefing can be invited to suggest how they can help with the requirements work with the objective of optimizing the efforts of all project members. A related need is to develop a strong sense of teamwork throughout the project. In my experience, an empowered and committed team can accomplish most anything it sets out to do.

Basic 12: Work proactively to address requirements-related risks. As is well known to practitioners, most projects do not address the risks they face with the discipline that they should. There are many excellent books concerning how to do this. A good starting point is Chapter 11 of [15]. Another source is [9], which provides a chapter on software requirements and risk management. Table 5 (see page 7) describes typical requirements-related risk response strategies that should be helpful.

Summary

From my experience in working with projects of all sizes, there is a lot of benefit to investing in and continuously improving the project's requirements process. There really is no good excuse for the extent of rework that is typically required on projects, or for the failure of projects, or even delivery of reduced functionality and/or over budget or beyond schedule issues. It is within our power to do better. Use of one or more of the requirements basics discussed here is likely to be helpful. Consider discussing the contents of this article at a project staff meeting with the objective of identifying the *top three* ideas, suggestions, or recommendations to improve the requirements practices of your project.◆

References

1. Whitten, Neal. Neal Whitten's No-Nonsense Advice for Successful Projects. Vienna, VA: Management Concepts, 2005.
2. Young, Ralph R. Effective Requirements Practices. Boston: Addison-Wesley, 2001.
3. Alexander, Ian F., and Richard Stevens. Writing Better Requirements. London: Addison-Wesley, 2002.
4. Gottesdiener, Ellen. Requirements By Collaboration. Reading, MA: Addison-Wesley, 2002.
5. Young, Ralph R. The Requirements Engineering Handbook. Boston: Artech House, 2004.
6. The Standish Group. "What Are Your Requirements?" West Yarmouth, MA: The Standish Group International, Inc., 2003.
7. Stevens, Richard, Peter Brook, Ken Jackson, and Stuart Arnold. Systems Engineering: Coping with Complexity. London: Pearson Education Limited, 1998.
8. Wiegers, Karl E. Peer Reviews in Software: A Practical Guide. Boston: Addison-Wesley, 2002.
9. Wiegers, Karl E. Software Requirements. 2nd ed. Redmond, WA: Microsoft Press, 2003 <www.processimpact.com/goodies.shtml>.
10. Hooks, Ivy F., and Kristin A. Farry. Customer-Centered Products: Creating Successful Products through Smart Requirements Management. New York: The American Management Association, 2001.
11. Young, Ralph R. "Recommended Requirements Gathering Practices." CrossTalk Apr. 2002 <www.stsc.hill.af.mil/crosstalk/2002/04>.
12. Leffingwell, Dean. "Calculating Your Return on Investment from More Effective Requirements Management." Rational Corporation, 1997 <www.128.ibm.com/developerworks/rational/library/347.html>.
13. Jones, Capers. Estimating Software Costs. New York: McGraw Hill, 1998.
14. Whitten, Neal. "Meet Minimum Requirements: Anything More Is Too Much." PM Network. Sept. 1998.
15. Young, Ralph R. "Project Requirements: A Guide to Best Practices." Vienna, VA: Management Concepts, 2006.
16. Sommerville, I., P. Sawyer, and S. Viller. "Viewpoints for Requirements Elicitation: A Practical Approach." Proc. of the 1998 International Conference on Requirements Engineering (ICRE '98), 6-10 Apr. 1998, Colorado Springs. IEEE Computer Society (1998): 74-81 <http://csdl2.computer.org/persagen/DLAbsToc.jsp?resourcePath=/dl/proceedings/&toc=comp/proceedings/icre/1998/8356/00/8356toc.xml>.

Notes

1. Personal e-mail to the author, September 2, 2006.
2. Request a briefing available on this topic from facilitator Charles Markert <markert@facilitationcenter.com>. For additional information, see <www.mid-atlanticfacilitators.net/>.
3. The term *real requirements* may be new to you. I refer to the requirements that are provided to developers by customers and users as the *stated requirements*. Industry experience has shown us that the stated requirements are *never* the real requirements for an application. Customers and users need help from the development team to *evolve* the real requirements from the stated requirements. This is one reason project managers should invest more time and money in the requirements process. It is also an area where projects can leverage the experience and expertise of *qualified* requirements analysts.
4. Capers Jones has reported that the U.S. average for requirements changes is about 2 percent per month during the design and coding phases, but can be much higher [12, p. 429]. Two percent per month equates to 24 percent per year; this is too much change in my experience to enable the project to remain in control.
5. See Neal Whitten's insightful and helpful article, "Meet Minimum Requirements: Anything More Is Too Much" [14] to gain an understanding of why it is in everybody's best interests to figure out the *minimum* set of requirements that will meet real needs. This article is reprinted with the author's permission as an appendix in [145].
6. The best guidance available on the important topic of traceability is James D. Palmer's article, "Traceability," originally published in *Software Requirements Engineering*, R.H. Thayer and M. Dorfman Eds. 1997, pp. 364-374. This article is reprinted with the author's permission as an appendix in [15].

About the Author

Ralph R. Young, Ph.D., teaches courses concerning requirements and process improvement and facilitates workshops to strengthen the use of practices and techniques on projects. He is the author of *Effective Requirements Practices*, *The Requirements Engineering Handbook*, *Project Requirements: A Guide to Best Practices*, and co-authored *Performance Based Earned Value.*

Northrop Grumman Information Technology Defense Group
7575 Colshire DR
McLean, VA 22102
Phone: (703) 556-1030
Fax: (703) 556-2802
E-mail: ralph.young@ngc.com

Interpreting Requirements in a He Said/She Said World

Deb Jacobs
Software Engineering Services

Interpreting what someone else really wants can be one of the most difficult elements of software development – at times it can be like talking to your teenager. That includes both the customer and the contractor. We must each walk in the other's shoes to see it from the other's vantage point. Requirements interpretation must be a two-way street with collaboration and communication the keys to success.

Burnt popcorn permeated the room as Mary walked down to Bob's cubicle. She couldn't believe that someone had burnt popcorn again; she was going to have to put instructions on the microwave since this was the third time this week someone had left it in too long. But right now she was too frustrated to even think about the popcorn for long. As she came to Bob's cubicle, she was starting to feel a little better about the calculations on the look-up report for the MDPSR database she was working on.

"Bob, do you have a minute?" she asked as she saw him huddled over his computer.

"Sure Mary, just a sec, while I finish my thought here." he replied.

As Mary sat down and waited she took another look at the words in the requirements matrix and drawings she had brought with her to show Bob. *I remember they talked a lot about what they wanted to see, but I'm not sure they decided anything,* she thought. Looking at the requirement again it just wasn't specific enough. It left a lot open to interpretation. *What was it they said about…?*

Bob surprised Mary a few minutes later when he said, "So Mary, what's up?" Mary jumped but recovered quickly. "Bob, I'm working on the look-up report and I'm not sure I understand what fields are used in this calculation."

Bob looked at the requirements matrix and Mary's notes and drawings.

"I remember this," said Bob. "They threw out several things at the requirements meeting a few months ago, but I can't remember what they finally decided or if they finally decided what they wanted."

"So what do you think I should do?" Mary asked. "I talked to Mike when I saw him in the hallway the other day and he told me which ones he used. Since he's one of the users, don't you think that should be right?"

"I don't know Mary, maybe we should ask Don," Bob suggested.

"Last time I asked him about a requirement he treated me like I was stupid or something. Besides I'm not sure we have time – this thing is scheduled for testing in 28 days, and I still have a lot to do," Mary exclaimed, exasperated.

"I know. He did that to me too and with this thing due soon we don't want to get burned again for not making the schedule. I can't take much more overtime – I'm getting burned out." Bob replied, "By the way, who burnt the popcorn this time?"

"I don't know, but it makes you never want to eat popcorn again in your life." Mary said as she walked away to finish her look-up report. She wondered if she was doing the right thing or if she should get clarification, but the time crunch brought her back to reality and she decided to follow the user's advice.

One month later at the meeting following testing, Mary knew that was the wrong decision. "Where did these figures come from?" Don ranted, as he went over the reports generated from the tests. "We told you what fields to use at the requirements meeting last April. In fact, there are several of these reports that aren't right. What's going on here?"

Mary grimaced, as Jack, the Project Manager (PM) – who had been named PM since he was good at placating people like Don – got up and started to explain that they had gone by the documented requirements. Before Jack could get two words out of his mouth, Don stood up and threw the reports on the table.

"This will all have to be fixed. I want to go to retest in one week."

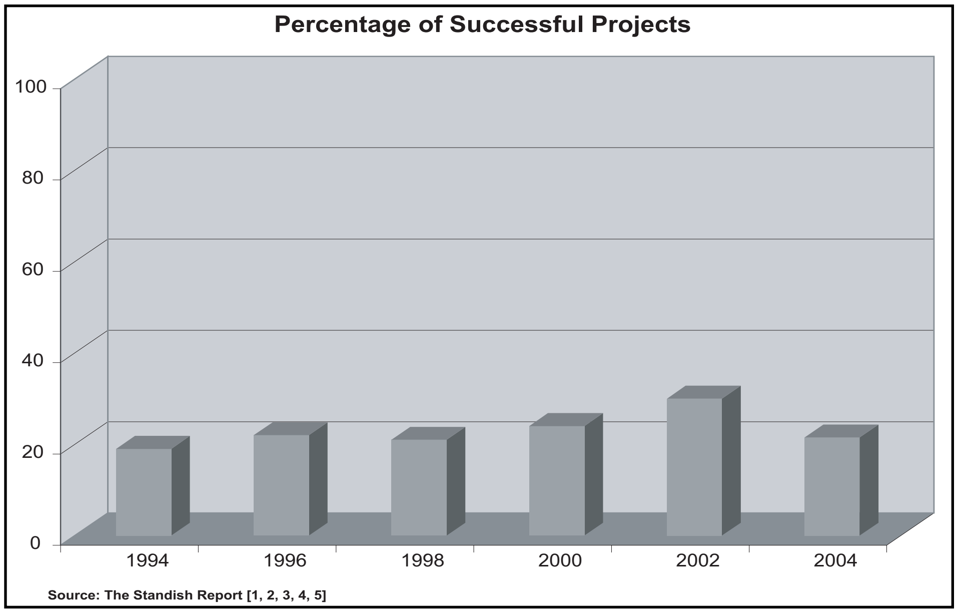
Sound familiar? Who is at fault here: the contractor receiving the requirements or the customer who gave the requirements? The answer is both. This scene may sound familiar to you since it happens time and again on project after project.

Regardless of which report you read, the battle cry is loud and clear: *Projects are failing more often than they succeed. Something must be done.* But what? That is the million-dollar question. Even the oft-quoted Standish Report still shows that successful projects that meet schedule, budget, and requirements still only hover around the 30 percent mark, as shown in Figure 1.

A Great Start

A great start for fixing this long-standing software development crisis is with requirements. We must fully understand what we are developing before we can develop the right product for our cus-

Figure 1: *Standish Report Summary*



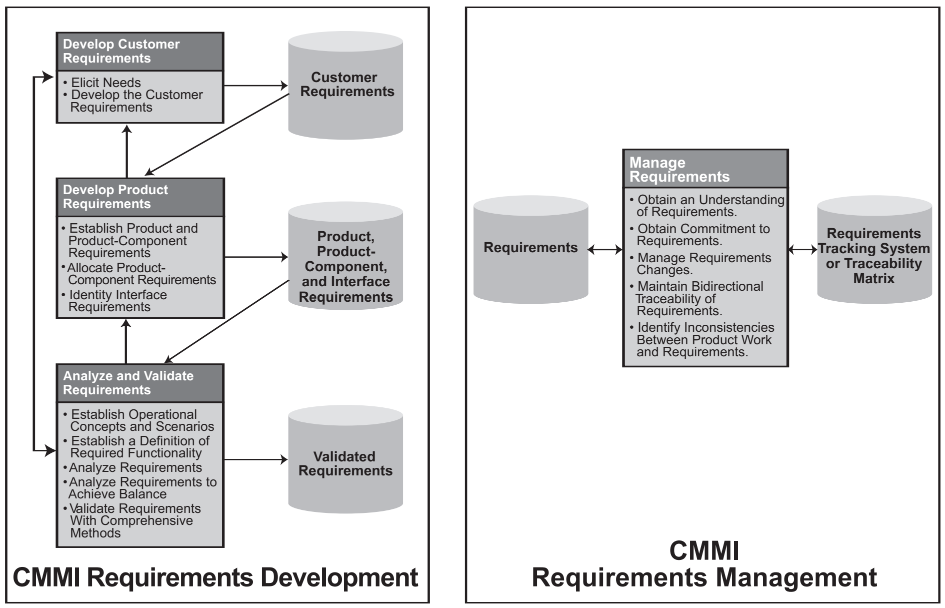

Develop Customer Requirements
• Elicit Needs
• Develop the Customer Requirements
Customer Requirements
Develop Product Requirements
• Establish Product and Product-Component Requirements
• Allocate Product-Component Requirements
• Identity Interface Requirements
Product, Product-Component, and Interface Requirements
Analyze and Validate Requirements
• Establish Operational Concepts and Scenarios
• Establish a Definition of Required Functionality
• Analyze Requirements
• Analyze Requirements to Achieve Balance
• Validate Requirements With Comprehensive Methods
Validated Requirements
CMMI Requirements Development
Requirements
Manage Requirements
• Obtain an Understanding of Requirements.
• Obtain Commitment to Requirements.
• Manage Requirements Changes.
• Maintain Bidirectional Traceability of Requirements.
• Identify Inconsistencies Between Product Work and Requirements.
Requirements Tracking System or Traceability Matrix
CMMI
Requirements Management

Figure 2: *Requirements Development and Requirements Management Process Areas* [7]

tomers. Readers will probably shake their head and say *of course we do*, but the statistics show that even though this seems to be a *no brainer*, we still are not doing it. This article will discuss some tried and true as well as some innovative methods that can help contractors and their customers through the quagmire of requirements elicitation.

There are many great sources for requirements engineering; I have provided some of the best sources at the end of this article. Based on my more than 25 years of experience in various software development and project management roles, I have found one of the best sources to be the Capability Maturity Model Integration^{SM} (CMMI$^{®}$) [6], developed by the Software Engineering Institute (SEI^{SM}). There are two process areas that are valuable resources for any project in defining and managing requirements: Requirements Development, and Requirements Management. Figure 2 illustrates these key process areas.

Eliciting Requirements

There are a myriad of methods for eliciting requirements. The key is interpreting the requirements correctly. The amount of time spent on eliciting requirements will depend on many factors such as team experience, management, level at which requirements have been pre-defined, and life-cycle methodology selected for development. The rule-of-thumb I have always used is approximately 15 percent of project time should be spent on identifying, defining, and clarifying requirements throughout the development life cycle with the majority of time spent up front. Some of the more popular methods for obtaining requirements include the following:

- Analysis of existing documentation.
- Statement of work/task definition.
- Interviews (structured and unstructured).
- Group brainstorming.
- Observation.
- Questionnaires and/or surveys.
- Prototyping.
- Modeling (enterprise, data, behavior, domain, non-functional).
- Rapid Application Development.
- Joint Application Development.
- Cognitive (examining usability).

A lot of resources have defined what a *good* requirement should look like. Most agree that it should be complete, consistent, unambiguous, verifiable/measurable, necessary, concise, implementation-free, and attainable. The additional reading at the end of this article provides great guidance for developing a *good* requirement.

There are some other methods that have been used successfully that are not quite as obvious as defining *good* requirements. These are the ones that this article will concentrate on.

Perspective Factor

Interpreting requirements correctly is the most prevalent problem in requirements engineering. We all look at things differently based on our background, education, experience, and simply from where we are standing at the moment. For requirements development, it will also depend on our responsibilities on the project.

MC Escher is famous for his optical illusion art (was and is well known for his impossible structures). A favorite is called *Relativity* that simply tells us that what you see is relative to where you are standing. When dealing with others realities, we have to see things from others' perspectives.

Each stakeholder is going to see something different. The customer or requirements giver sees the final products based on their outlook of either the manual methods used or a legacy system they have been using. Many times it is hard to articulate either in writing or verbally what they want even when they know exactly what they want. The contractor or requirements receiver sees the same thing based on their experiences in development, using like systems, etc. By looking at things from each others vantage point, we are able to better understand what needs to be developed. It is everyone's responsibility.

Proactive Requirements Management

Planning requirements management activities, open communications, and proactive resolution of even a simple *not sure* is crucial on a project in order to avoid nightmare scenarios later on. There is enough to panic about when developing a system without the added stress of misunderstandings or misconceptions.

Planning the requirements development activities can alleviate much of the typical *Keystone Kops* scenarios. Lack of planning has been oft-quoted as a top cause for failure in survey after survey. Lewis Carroll in *Alice in Wonderland* said,

> It sounded an excellent plan, no doubt, and very neatly and simply arranged. The only difficulty was, she had not the smallest idea how to set about it. [8]

More important than the planning process is ensuring the plan does not become shelfware: it must be useable and thus used by the project team. The level of planning will depend upon the size and scope of the project. Regardless of the project, a key word in planning is *flexibility*. Even the most closely planned activities can be unsuccessful without flexibility to accommodate unforeseen events and changing priorities.

Working closely with the customer throughout the development process can make the difference between success and failure. The key is for the customer to help ensure that the system's desired function-

® Capability Maturity Model and CMMI are registered in the U.S. Patent and Trademark Office by Carnegie Mellon University.

SM SEI is a service mark of Carnegie Mellon University.

ality is realized and the developer fully understands what the system's desired functionality is from the customer's point of view. The ever-growing-in-popularity Agile methodologies uses *active stakeholder participation* and *on-site customer* to describe this customer/contractor collaboration.

Who's Who

A graphical depiction for defining and keeping track of who's who provides a great tool for both the customer and the contractor. This ensures that there are no misunderstandings or misconceptions. For larger projects with numerous stakeholders, including subcontractors, this tool is essential in providing a means of averting potential issues resulting from wrong requirements.

So who is the best person to act as the decision maker? The customer (administrative or technical)? The contractor? The subcontractor? Or someone else? Typically, it is a combination of the administrative customer and the technical customer representatives. However, it depends upon the contract. Many times the contractor has the pertinent skills and knowledge to make the appropriate decisions concerning requirements so they may be tasked as the decision maker. Ultimately, the customer usually has the largest stake in the outcome of the project since they have to live with the results. They should be involved with the requirements decisions at all times.

In short, the key is to know the *who*, *what*, *when*, *where*, *why*, and *how*. There should be a clear understanding of who are the decision makers for requirements. The *who's who* diagrams should be approved by the appropriate managers and distributed to the entire team. If a project has a communications plan, this is a good place to include these diagrams and should include who can accomplish the following:

- Add a new requirement.
- Change existing requirements.
- Clarify a requirement.
- Accept changes to requirements.
- Direct the various teams.
- Determine if a requirement has or has not been met.
- Accept requirements as met or not met.

Distribution lists for the various project elements should be developed to ensure all stakeholders are included as needed.

Win-Win Stakeholder Negotiations

A process should be in place that is fair to both the customer and the contractor, thus *win-win*. The customer is responsible for ensuring they have conveyed the correct requirements, and the contractor is responsible for ensuring they understand the requirements correctly, and an approved requirements method can be developed. Some of the most effective methods of documenting approval and tracking requirements are the use of requirements matrices and requirements databases depending upon the size of the effort. Whichever method is used, the key is *approved*.

An effective method of providing a win-win is through the use of win-win stakeholder negotiations. These negotiations can be in the form of meetings that provide a forum for open discussions between stakeholders at the worker-bee level (the engineers developing the systems and the ultimate users of the system). Requirements can be negotiated based on technology, environment, time, effort, and budget constraints. They provide a way of fully understanding the contractual requirements and discussing derived requirements. Someone with the power to make decisions on the project must be present in order to make the meetings effective.

"A process should be in place that is fair to both the customer and the contractor, thus win-win."

Sometimes a trained negotiator can help alleviate some of the pain involved in these meetings. Role-playing is a great method used where the customer and contractor trade places and look at things from each others perspective. The goal is for everyone to walk away satisfied, or at least *break-even*.

Many times, either the contractor or the customer will have tasks that need to be accomplished as part of the requirements elicitation; the negotiations meetings can include discussion and negotiation of these commitments for both the customer and the contractor. To make adherence to the agreed-upon commitments more effective, some organizations use what they call a stakeholder's contract that puts these particular commitments in writing.

Test Early, Test Often

Test early, test often. This should be the mantra for all projects. If we think of a project in terms of how we can test it, we will more readily be able to see what needs to be developed. This should again be a collaborative effort between the requirements giver and receiver.

On one project in which I was staff engineer, I was responsible for a replanning effort since the project was doomed to go over both schedule and budget. After much discussion, we decided to try an incremental approach where requirements were *chunked* into categories and the system was built incrementally. Starting with the system communications as the foundation, we tested this component and then built requirements into a solid framework. This allowed us to avoid the inherent problem of finding bugs when a large system is fully integrated. By working closely with the client, it also avoided finger-pointing at the end of the project since they were involved throughout the testing and the requirements were fully fleshed out. This project finished within the original schedule and budget.

If an incremental approach does not work or the project is not large enough for that level of testing, early development of test cases, use cases, or test scenarios can enable the developer to more easily visualize the finished product. It provides a chance to work with the customer to ensure a full understanding of the requirements. It also weeds out or helps clarify requirements that are not testable or in some way measurable.

Another proven method of fleshing out requirements is prototyping to simulate the final product or a complex component of the final product. There are many benefits to be gleaned from prototyping, including exposure of misunderstandings between customers and developers, detection of missing functionality or services, identification of confusing functionality or services, development of a simplified working system early in development cycle for the user, incremental delivery of a system starting with the prototype, identification of risks, and a basis for derivation of requirements.

If a project can afford the extra time and expense of prototyping or there are safety or security issues involved, prototyping is key to building a successful system. Depending upon the system under development, the time and expense of doing a prototype may actually cost less in time and ultimately be less expensive. There are many factors that must be weighed in making a decision to prototype. Sometimes a simple paper prototype makes the difference between day and night.

The Value of Pictures

Conceptual modeling provides a great method for visualizing the final system. Fred R. Barnard said, "One picture is worth a thousand words [9]." Use paper, use a whiteboard, or use graphics applications, but by all means draw pictures. Draw as many pictures as it takes to fully understand and agree on each requirement. Some of the most successful projects are accomplished by spending as much time in front of the white board as in front of the computer. This is by far the easiest and best method for interpreting and agreeing upon requirements. It can also be effective in weeding out requirements not considered, identifying potential or real bottlenecks, or deriving requirements. Some effective popular methods of graphically depicting requirements include the following:

- Data-flow diagrams.
- Flowcharting.
- Cross-functional flowchart (charted by responsibility).
- Information mapping.
- Entity relationship diagram.
- Unified Modeling Language (UML) use case.
- UML collaboration/communications diagram.
- UML state chart diagram.
- UML sequence diagram.
- UML activity chart.
- UML component diagram.
- UML deployment diagram.
- Structured analysis and structured design.
- Structured analysis and design technique.
- Integrated computer-aided manufacturing definition family of methods.
- State transition diagram.
- Object role modeling.
- Decision trees.
- Role activity diagrams.
- Petri net.

There are numerous methods for helping developers and customers to visualize the final product. The use of several of these diagrams is optimal when interpreting requirements in order to fully understand and communicate. They can be used with various levels of detail depending upon what the person is trying to convey. I have found that by keeping each diagram/picture as simple as possible, the greatest value can be derived so remember the *KISS* principle: Keep it short and simple. The level of detail in the initial diagrams should be flexible enough to leave room for analysis and optimization. There are a number of tools available to make using these methods easier.

Drilling Down/Chunking

Drilling down and chunking are ways of examining each requirement for full understanding. Drill down can be used to decompose requirements by starting at the basic high-level requirement and drilling down to the details of each requirement. After the high-level requirements are fully understood and agreed upon, the development team should move to fine-tune the details. Drill down should be iterative; as more details of higher-level requirements are understood, they are drilled down to lower levels for a complete understanding of what the customer is looking for. This is where the derived and implied requirements are defined. This should be accomplished until all requirements are fully fleshed out.

Drill down ensures that time and money is not wasted on detailing requirements that are misunderstood from the beginning. The decomposition must be approved in order to move forward to the next iteration, thus avoiding further misunderstandings. This step can be used to associate each requirement with a particular subsystem or component. It can also include verification/testing strategy and method generation.

Chunking is a method used to organize and arrange information so that it is easier to read, understand, access, and retrieve. It is a method of subdividing and organizing into short *chunks* of information in a uniform format. The simpler the better is key for certain types of information, with clutter being the villain. The premise is to group information and provide white space to break information into manageable components. This is sometimes called *visual chunking* and results in greater readability and accessibility. Chunking can be used effectively in understanding requirements especially if they are obtained from a statement of work or other typical task definition document. When you look at elements in logical groupings, you are able to remember and understand them much easier. There are things to avoid with chunking, including over-generalization and being over-specific. Keep it simple, but do not take away from content and run the risk of not being fully understood.

Balancing Act

One of the most difficult things encountered during requirements definition is balancing the customers and, ultimately, the user's needs versus their expectations. The needs are the identified requirements, which many times differ from their actual expectations, which are the unidentified requirements. Sometimes understanding their expectations can drive the understanding of the needs for defined requirements and gain insight into the *real* requirements. This can be accomplished by spending some time with the users, but with the caution that the designated decision maker is the one with the authority to add, change, delete, or alter the defined requirements in any way. It is easy for developers to get caught up in the *nice to have's* or *this is the way I do it* from users.

The second – and perhaps more important – balancing act is typically called *managing requirements creep*. Getting feedback from the customer is key, but at the same time a close eye must be kept on out-of-scope requirements that drive the project: schedule, cost, and risk. Flexibility is important to ensure that all the customer's requirements are met, which is the ultimate goal of any development project. In fact, for the Agile programmer, changing requirements are a way of life; they are expected and embraced. But the key is to balance these changing requirements with feasibility, applicability, and impactability to the system under development.

Iterative Requirements Interpretation

Interpreting requirements means that each requirement must be examined to ensure a full understanding. This is an iterative process of meetings for interviewing or brainstorming until all of the requirements have been fleshed out and beyond. Where did meetings get such a bad reputation? Used properly, meetings are a great tool for ensuring everyone is dancing to the same beat.

Many of the Agile methodologies use meetings on a regular basis. These meetings are called *stand-ups* or *scrums*. They are short, quick meetings first thing every morning to briefly discuss what actions are in progress. These meetings go a long way in ensuring no misunderstandings or misconceptions throughout the development process. The key is to make the meetings applicable and quick. They must include the appropriate project stakeholders. If side issues need to be resolved, this should be done in a separate meeting with only the necessary stakeholders. Remember that there are designated decision makers who must be kept involved or at least informed at each step, especially when requirements change.

Summary

By always looking at things from each others vantage point, the software development world can start to overcome some of the nightmare scenarios so typical of software development projects. Most of these nightmares are directly attributable to requirements and the understanding of each requirement. By using some of the techniques discussed in this article, the typical Mary and Bob scenario can be avoided. Heed the battle cry: We can make projects more successful if we all work together. When all project stakeholders work from the perspective that this is a collaborative effort, then everyone wins.◆

References

1. The Standish Group. "CHAOS: A Recipe For Success, 1998." Standish Group International, 1999.
2. The Standish Group. "CHAOS Reports." Standish Group International <www.standishgroup.com>.
3. The Standish Group. "The Standish Group Report – CHAOS 1994." Standish Group International, 1995.
4. The Standish Group. "What Are Your Requirements?" Standish Group International, 2003.
5. Hayes, Frank. "Chaos Is Back." Computerworld Nov. 2004.
6. Carnegie Mellon University. "CMMI^{SM} for Systems Engineering/Software Engineering, Vers. 1.1, Staged Representation." CMU/SEI-2002-TR-002. Dec. 2001.
7. Jacobs, Deb. "Accelerating Process Improvement Using Agile Techniques." Auerbach Publications, 2006.
8. Carroll, Lewis. Alice's Adventures in Wonderland and Through the Looking Glass. Reissue edition. Signet Classics, 2000
9. Barnard, Frederick R. Printers' Ink. Mar. 1927.

Additional Reading

1. Bashar Nuseibeh, and Steve Easterbrook. "Requirements Engineering: A Road Map." 3rd International Symposium on Requirements Engineering <www.cs.toronto.edu/~sme/papers/2000/ICSE2000.pdf>.
2. Scott Ambler. Ronin International, Inc. The Elements of UML Style <www.modelingstyle.info>.
3. McConnell, Steve. Construx Software <www.stevemcconnell.com> or <www.construx.com>.

About the Author

Deb Jacobs has more than 28 years of experience in information technology, including system and software engineering, project management, process improvement and proposal development and has helped many organizations be more successful in development and management. She has provided CMMI expertise to numerous organizations, including training, implementation, and assessments/appraisals. Jacobs is former *Spinout* newsletter editor/originator, former Computer Engineering Readiness Team conference chairperson, Infotec Deputy Software Tracks Chair, SEI CMMI contributor, and is a member of the CrossTalk Editorial Board. Jacobs has authored several technical articles and the popular process improvement book *Accelerating Process Improvement Using Agile Techniques*. She has a bachelor of science in computer science.

Software Engineering Services
1508 JF Kennedy DR
STE 100
Bellevue, NE 68005
Phone: (402) 932-5349 or
(402) 292-8660
E-mail: djacobsfpa@aol.com or
djacobs@sessolutions.com

Coming Events

January 3-6
HICSS-40
Hawaii International Conference on System Sciences
Waikoloa, HI
/www.hicss.hawaii.edu/hicss_40/apahome40.htm

January 7-12
COMSWARE 2007
Second International Conference in Communication System Software and Middleware
Bangalore, India
www.comsware.org

January 8-12
CBIS '07 Chemical and Biological Information Systems Conference and Exhibition
Austin, TX
www.ndia.org/

January 17-19
CEA '07 Computer Engineering and Applications
Queensland, Australia
http://wseas.org/conferences/2007/australia/cea/

January 29-31
Enterprise Architecture Practitioners Conference
San Diego, CA
www.opengroup.org/sandiego2007

June 18-21, 2007
2007 Systems and Software Technology Conference

STC Systems & Software Technology Conference

Tampa, FL
www.sstc-online.org

Letter to the Editor

Dear CrossTalk Editor,

I just wanted to say that those were REALLY great articles (very detailed, nicely written). I got into *Star Wars* this summer, and I know how exciting it can be. See if you can beat this: I've watched every episode except *Revenge of the Sith*, I'm going to be Jango Fett the bounty hunter for Halloween, AND I'm going to be Luke Skywalker when my school has Spirit Week!

May the Force be with you.

— Katelyn Crombie, age 12

Experiences in Eliciting Security Requirements

Dr. Nancy R. Mead
CERT, Software Engineering Institute

There are many requirements elicitation methods, but we seldom see elicitation performed specifically for security requirements. One reason for this is that few elicitation methods are specifically directed at security requirements. Another factor is that organizations seldom address security requirements elicitation specifically and instead lump them in with other traditional requirements elicitation methods. This article describes an approach for doing trade-off analysis among requirements elicitation methods. Several case studies were conducted in security requirements elicitation; the detailed results of one case study and brief results of two other case studies are presented here.

A largely neglected area in requirements engineering is that of elicitation methods for security requirements. Many organizations, if they use an elicitation method at all for security requirements, use one that they have previously used for ordinary, functional (end-user) requirements. Alternatively, they may decide to use a brainstorming approach. Such methods are usually not oriented towards security requirements and do not result in a consistent and complete set of security requirements.

Carnegie Mellon University (CMU) graduate students, working with me, selected and applied several elicitation methods in a series of case studies [1]. In this article, I describe a trade-off analysis that we used to select a suitable requirements elicitation method and present results detailed from a case study of one method and a series of two other methods, used in a series of case studies. While results may vary from one organization to another, the discussion of our selection process and the example results should apply to all.

Elicitation Methods

The following is a sample of methods that could be considered for eliciting security requirements.

Misuse Cases

Misuse cases apply the concept of a negative scenario – that is, a situation that the system's owner does *not* want to occur – in a use-case context. For example, business leaders, military planners, and game players are familiar with analyzing their opponents' best moves as identifiable threats. By contrast, use cases generally describe behavior that the system or entity owner wants the system to show [2]. Use-case diagrams have proven quite helpful for the elicitation of requirements.

Soft Systems Methodology (SSM)

SSM deals with problem situations in which there is a high social, political, and human activity component [3]. The SSM can deal with *soft problems* that are difficult to define, rather than *hard problems* that are more technology oriented. Examples of *soft* problems are how to deal with homelessness, how to manage disaster planning, and how to improve Medicare. Eventually, technology-oriented problems may emerge from these *soft* problems, but much more analysis is needed to get to that point.

Quality Function Deployment (QFD)

QFD is an overall concept that provides a means of translating customer requirements into the appropriate technical requirements for each stage of product development and production [4]. The distinguishing attribute of QFD is the focus on customer needs throughout all product development activities. By using QFD, organizations can promote teamwork, prioritize action items, define clear objectives, and reduce development time.

Controlled Requirements Expression (CORE)

CORE [5, 6] is a requirements-analysis and specification method that clarifies the user's view of the services to be supplied by the proposed system. In CORE, the requirements specification is created by the user and the developer, not one or the other. The problem to be analyzed is defined and broken down into user and developer viewpoints. Information about the combined set of viewpoints is then analyzed. The last step of CORE deals with constraints analysis such as the limitations imposed by that system's operational environment in conjunction with some degree of performance and reliability investigation.

Issue-Based Information Systems (IBIS)

IBIS, developed by Horst Rittel, is based on the principle that the design process for complex problems, which Rittel terms *wicked* problems, is essentially an exchange among the stakeholders in which they bring their personal expertise and perspective to the resolution of design issues [7]. Any problem, concern, or question can be an issue and may require discussion and resolution in order for the design to proceed. The IBIS model centers on the discussion and resolution that is an integral part of the design process.

Joint Application Development (JAD)

JAD [8] is specifically designed for the development of large computer systems. The goal of JAD is to involve *all* stakeholders in the design phase of the product via highly structured and focused meetings. In the preliminary phases of JAD, the requirements-engineering team is tasked with fact finding and information gathering. Typically, the outputs of this phase – as applied to security requirements elicitation – are security goals and artifacts. The actual JAD session is then used to validate this information by establishing an agreed-upon set of security requirements for the product.

Feature-Oriented Domain Analysis (FODA)

FODA is a domain analysis and engineering method that focuses on developing reusable assets [9]. By examining related software systems and the underlying theory of the class of systems they represent, domain analysis can provide a generic description of the requirements of that class of systems in the form of a domain model, and a set of approaches for their implementation.

The FODA method was founded on two modeling concepts: abstraction and refinement. Abstraction is used to create domain models, as described above, from the specific applications in the domain. Specific applications in the domain are developed as refinements of the domain models. The example domain used in the report [9] is that of window-management systems.

Critical Discourse Analysis (CDA)

CDA uses sociolinguistic methods to analyze verbal and written discourse [10]. In particular, CDA can be used to analyze requirements elicitation interviews and to understand the narratives and *stories* that emerge during requirements elicitation interviews.

Accelerated Requirements Method (ARM)

ARM is a facilitated requirements elicitation and description activity process [11]. Overall, there are three phases of the process: Preparation phase, Facilitated Session phase, and Deliverable Closure phase. The ARM process is similar to JAD, but it also has certain significant differences with respect to the baseline JAD method. In the ARM process, the facilitators are content-neutral, the group dynamic techniques used are different from those used in JAD, the brainstorming techniques used are different, and the requirements are recorded and organized using different conceptual models.

Evaluation Criteria

The following are example evaluation criteria (project participants need to have a common understanding of what they mean in order to use them in selecting an elicitation method):

- **Adaptability.** The method can be used to generate requirements in multiple environments. For example, the elicitation method works equally as well with a software package that is near completion as with a project in the planning stages.
- **Computer-aided software engineering (CASE) tool.** The method includes a CASE tool. (The Software Engineering Institute [SEI] defines a CASE tool as *a computer-based product aimed at supporting one or more software engineering activities within a software development process* [1].)
- **Stakeholder acceptance.** The stakeholders are likely to agree on the elicitation method in analyzing their requirements. For example, the method is not too invasive in a business environment and can be implemented in a reasonable amount of time.
- **Easy implementation.** The elicitation method is not overly complex and can be properly executed easily.
- **Graphical output.** The method produces readily understandable visual artifacts.
- **Quick implementation.** The requirements engineers and stakeholders can fully execute the elicitation method in a reasonable length of time.
- **Shallow learning curve.** The requirements engineers and stakeholders can fully comprehend the elicitation method within a reasonable length of time.
- **High maturity.** The elicitation method has experienced considerable exposure and analysis in its vetting by the requirements engineering community.
- **Scalability.** The method can be used to elicit the requirements of projects of different sizes, from enterprise-level systems to small-scale applications.

Ranking Against the Criteria

The elicitation methods can be ranked against the criteria using a tabular form. In Table 1, we have filled in the values that the student team decided on for the sample methods. Each method was rated according to the desired features, and the rankings were simply added to provide a summary result. A weighted average could also have been used if some features were considered to be more important than others. For example, availability of a CASE tool might be more important than graphical output. A typical weighting scheme could consider criteria to be *essential* with weight 3, *desirable* with weight 2, and *optional* with weight 1. This sort of evaluation is subjective, particularly since the students worked under time constraints and did not have prior experience with this, so results may vary. Each organization or project should develop its own comparison criteria and its own ratings.

	Misuse Cases	SSM	QFD	CORE	IBIS	JAD	FODA	CDA	ARM
Adaptability	3	1	3	2	2	3	2	1	2
CASE Tool	1	2	1	1	3	2	1	1	1
Stakeholder Acceptance	2	2	2	2	3	2	1	3	3
Easy Implementation	2	2	1	2	3	2	1	1	2
Graphical Output	2	2	1	1	2	1	2	2	3
Quick Implementation	2	2	1	1	2	1	2	2	3
Shallow Learning Curve	3	1	2	1	3	2	1	1	1
High Maturity	2	3	3	3	2	3	2	2	1
Scalability	1	3	3	3	2	3	2	1	2
Total Score	**18**	**18**	**17**	**16**	**22**	**19**	**14**	**14**	**18**

3 = Very Good, 2 = Fair, 1 = Poor

Table 1: *Comparison of Elicitation Methods*

In our case studies, we decided to use JAD, ARM, and IBIS on three different projects. These three methods were subjectively ranked to be the most suitable candidates for the case studies, given our constraints. The student team's allotted time was constrained, since this was a one-semester project. It was also the case, however, that the clients had limited time to devote to this exercise. Therefore, time constraints are mentioned several times.

It is important to note that although students did the elicitation, the projects studied were real industry and government projects, not software projects developed by students in an academic setting. It is also possible that a combination of methods may work best. This should be considered as part of the evaluation process.

Security Requirements Elicitation Results

In this section, we present brief results for IBIS and JAD and detailed results for ARM. Detailed results for all three methods can be found in the Requirements Engineering section of the BuildSecurityIn Web site [12] and in the case study report [1].

Brief Results for IBIS

The effectiveness of IBIS in eliciting security requirements depends on the quality of the interview questions. To the greatest extent possible, the scope of questions must cover the entire range of security requirements that could possibly involve the system. We found that the interviewer must be persistent in encouraging the stakeholders to explain their rationale when proposing a solution to an issue. By explaining why they have chosen such a position, the stakeholders can naturally discuss the pros and cons among themselves. In addition to proper question selection, we found that the success of IBIS is directly proportional to the variety and level of participation of stakeholders in the project. In fact, in our case study, IBIS worked best when different stakeholders presented opposing viewpoints, which is common in large-scale projects. The Compendium software tool associated with IBIS was easy for the student team to use and effective in generating IBIS maps. To avoid displaying extremely large maps (which our stakeholders found difficult to read), we recommend exploiting the nested maps feature in Compendium. This feature enables the user to hide some of the lower-level details of the maps by nesting them inside other map elements, while maintaining the ability to drill down into the details if requested. In fact, a comment received from the stakeholders indicated that such a hierarchical map structure would have been more beneficial in handling some of the larger maps.

Brief Results for JAD

Due to time constraints, we did not define the work flow, data elements, screens, and reports of the project, so the JAD method turned out to be very similar to an unstructured interview process. Although unstructured interviews were used in an earlier case study, we did not attempt to do a direct comparison of the JAD results with those earlier case study results. In essence, the team just asked the stakeholders some questions about the project. Thus, the team did not use the full capability of the JAD method, which may have biased the results. The JAD session phase was designed for developing functional (end user) requirements; there was no specific way to discuss quality requirements such as security. Therefore, the team spent a lot of time researching other methods to assist in obtaining better security requirements during the JAD session. The team suggests that JAD be used with an additional method to deal with quality requirements.

Detailed Results for ARM

Results obtained using ARM on a government project are described below. This is not a military project, but the security concerns, such as access control, are similar to the typical security concerns of military projects. ARM is designed to elicit, categorize, and prioritize security requirements. As noted earlier, the ARM method includes three phases.

Preparation Phase

As the name implies, this phase is used to prepare for the Session phase. There are six steps in the Preparation phase:

1. Define goals, objectives, and project success criteria (PSC) of the project.
2. Define objectives and preliminary scope of the session.
3. Establish partitions and identify participants.
4. Determine environmental and logistical aspects.
5. Establish expectations for participants.
6. Communicate with participants.

One way to obtain this information is to use a feedback form composed of questions for the stakeholders. The list of questions can be found in the case study report [1]. The stakeholders should be given a few business days to complete and return the form. In the meantime, the requirements engineering team can prepare a memorandum containing goals, objectives, PSC, preliminary scope, partition definitions, participants, and logistic arrangements. Participants should read the memorandum before the Session phase to understand the content, expectation, and goals of the method. The overall goal of the memorandum is to increase the quality of the Session phase.

"Results obtained using ARM on a government project are described here. This is not a military project, but the security concerns, such as access control, are similar to the typical security concerns of military projects."

Depending on the results of the stakeholders' feedback forms, another meeting with the stakeholders may be necessary before beginning the Session phase.

Session Phase

The Session phase is the heart of the ARM process. It includes six steps:

1. Executive sponsor commentary.
2. Scope closure.
3. Brainstorm, organize, and name (BON).
4. Details.
5. Prioritization.
6. Participant feedback.

Before the Session phase meeting, logistical arrangements should be made to ensure that the meeting goes smoothly. The detailed list of logistical items can be found in the case study report [1].

Executive Sponsor Commentary. This step allows executive sponsors to provide introductory remarks to the participants regarding the planned session. Depending on the project organization, this step may or may not be necessary.

Scope Closure. The purpose of this step is to define what is in or out of scope. When eliciting security requirements, participants might need to familiarize themselves with security issues ahead of time to make this determination.

BON. The BON step provides an efficient way to elicit the candidate requirements from participants. The requirements engineering team can start by asking the participants the *focus question*, which should be crafted to tie to the goals, objectives, and scope of the project together. For example: *An important security requirement of the beta application is…* Based on their professional experience and security knowledge, the participants are then asked to write down seven important security requirements within seven minutes.

Next, the participants are asked to write their top three or four security requirements on cards within three minutes. The requirements engineering team then collects the cards and displays the candidate security requirements. The candidate security requirements produced in this example are listed in the following 24 initial requirements produced in ARM:

1. The ability to securely transmit data to remote sources.
2. The preservation of data integrity.
3. The enforcement and usability of an access control system.
4. Manageable security (and not hinder business where possible).
5. A strong, reliable authentication process.
6. Private information (from the outside world).
7. Consistent application program interfaces (APIs).
8. Data integrity.
9. Authentication and access control.
10. Strong authentication.
11. Risk reduction or elimination of risk of inappropriate behavior.
12. Granular access to data for users (operators) and customers.
13. Accountability (who did what, when, how...).
14. Integrity (assurance in data protection and validity).
15. Indelibility (deletions and retractions are noted/logged).
16. Integrity.
17. Access control.

18. Confidentiality (encryption, etc.).
19. Partitioned data store (public read only and private read/write).
20. Selectively secure communication with outside entities.
21. Segmented disclosure representation and support.
22. Role-based restricted views/edit/ action access (e.g., summary report information, public for particular people).
23. 24/7 availability via remote authenticated access and secure.
24. Key action audit (e.g., attribution of who/from where the publish button was pressed, what changes were made).

In the Organize step, all the participants review the candidate security requirements generated during the brainstorming session to see whether any duplicate or inadequate security requirements were included. Then the participants discuss what they think are important requirements. This step provides an opportunity for the participants to share their security concerns about the project. After a period of discussion and debate, they delete candidate security requirements that are viewed as redundant or inappropriate.

The participants removed requirements 1, 2, 5, 9, 14, 16, and 17. The remaining requirements after initial eliminations are:

3. The enforcement and usability of an access control system.
4. Manageable security (and not hinder business where possible).
6. Private information (from the outside world).
7. Consistent APIs.
8. Data integrity.
10. Strong authentication.
11. Risk reduction or elimination of risk of inappropriate behavior.
12. Granular access to data for users (operators) and customers.
13. Accountability (who did what, when, how...).
15. Indelibility (deletions and retractions are noted/logged).
18. Confidentiality (encryption, etc.).
19. Partitioned data store (public read only and private read/write).
20. Selectively secure communication with outside entities.
21. Segmented disclosure representation and support.
22. Role-based restricted views/edit/ action access (e.g., summary report information, public for particular people).
23. 24/7 availability via remote authenticated access and secure.
24. Key action audit (e.g., attribution of who/from where the publish button was pressed, what changes were made).

In the Name step, the participants are instructed to group the selected security requirements and create names for each group. In this example, security requirements, groups, and names were generated together and descend in order of importance from A-F (as shown in Figure 1, page 18). The security requirements are categorized into six groups, each containing between one and four security requirements. This step can result in addition or deletion of requirements. The following are the grouped requirements contained in each:

- **Group A: Confidentiality:** Information must be kept private from the outside world; communication with outside entities must be selectively secured.
- **Group B: Access Control:** Role-based restricted views/edit/action access (e.g., summary report information, public for particular people); enforcement and usability of an access control system; granular access to data for users (operators) and customers; segmented disclosure support and representation.
- **Group C: Data Integrity:** Partitioned data store (public read only and private read/write); indelibility.
- **Group D: Manageability:** Accountability; key action audit (e.g., attribution of who/from where the publish button was pressed and what changes were made); auditing capabilities.
- **Group E: Usability:** Security must be manageable and not hinder business (where possible); must be available 24/7 via remote authenticated access; must have consistent APIs; must reduce or eliminate risks of inadvertent behavior.
- **Group F: Authentication:** Strong authentication.

Details: Benefits, Proof, Assumptions, Issues, and Action Items. In Step 4, the participants are asked to evaluate each requirement using the following 10 questions:

1. Is the candidate requirement a fragment or duplicate of anything that has already been discussed?
2. According to the contributor and the group, is the candidate requirement fragment in scope?
3. Would you like to change the title?
4. If you had this capability, how would it help the business?
5. What will you consider acceptable evidence that the envisioned capability

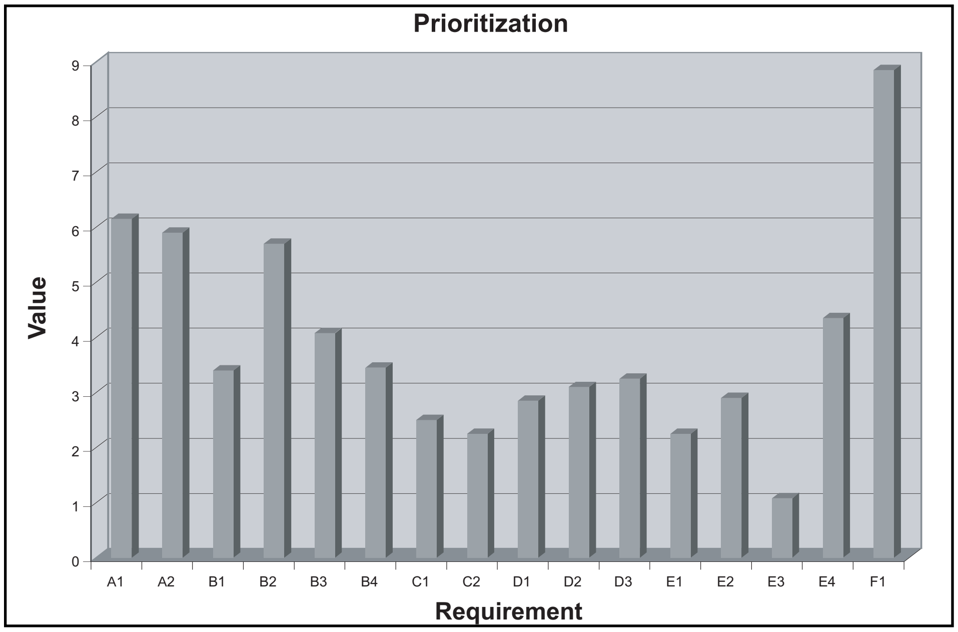


Figure 1: *Ranked Score of the Requirements*

has been successfully delivered to the business?
6. Are there any special constraints on the requirement?
7. Are there any assumptions made regarding the requirement?
8. What are the remaining issues and actions items for the requirement?
9. Are there any related notes or comments?
10. Is there anything that needs to be clarified by the supplier of the requirement?

In this case, the security requirements were reviewed collectively, not individually.

Prioritization. In the BON step of ARM, the participants generate the candidate security requirements of their project, then modify and refine the projected security requirements in the Details step to ensure that the requirements are unambiguous, clear, and concise.

The Prioritization phase of the ARM method begins with the requirements engineering team providing instructions to guide participants to label each requirement as either *A*, *B*, or *C*, where *A* stands for most important, *B* stands for very important, and *C* stands for important. The rankings are to be assigned equally across the security requirements.

After the session concludes, the scores are calculated. First, the requirements engineering team substitutes the rankings *A*, *B*, and *C* with numeric values 9, 3, and 1, respectively. Then, the team calculates the average score of each requirement. The results are shown in Figure 1.

These are the final requirements in priority order:

1. The system shall utilize cryptographically strong authentication.
2. The information in the system must be kept private from unauthorized users.
3. The system shall implement selectively secure communication with outside entities.
4. The system shall utilize and enforce an access control system.
5. The system will attempt to reduce or eliminate risks of inadvertent behavior.
6. The system shall provide granular access to data for users (operators) and customers.
7. The system shall provide role-based, restricted view, edit, and action access (e.g., summary report information, public for particular people).
   (tied with)
   The system shall represent and support segmented disclosure.
8. The system shall implement auditing capabilities.
9. The system shall provide accountability of users' actions.
   (tied with)
   The system will be available 24/7 via remote authenticated access.
10. The system shall maintain a partitioned data store that is public read only and private read/write.
    (tied with)
    The system shall implement a key action audit (e.g., attribution of who pressed the publish button and from where, and what changes were made).
11. The system shall implement indelibility.
    (tied with)
    Where possible, the system's security features must be manageable and not hinder business.
12. The system shall expose consistent APIs to developers.

Based on the result of the prioritization, the participants can then develop a plan to effectively implement their security requirements.

Stakeholders' Feedback. In the final portion of the Session phase, the team requested that the participants fill out a feedback form that was used to collect information to improve the method. Example questions are similar to the following:

- What did you like or not like about the Session phase?
- What did you think was the most important part of the Session phase?
- What would you change about the Session phase?

Deliverable Closure

In this study, the set of stakeholders was relatively small and Deliverable closure took place informally at the Session phase.

Results Summary for ARM

Overall, ARM is an effective and rapid method of collecting requirements. By simply choosing the correct focus question, the process is easily adapted to elicit security requirements. Due to the large number of questions that must be asked for each requirement, we recommend enforcement of strict time management and proactive guidance of the discussions among the stakeholders.

Depending on the security expertise of the participants, the requirements engineering team may need to review some security concepts with the participants before the session begins.

ARM was developed for use in a commercial environment, and thus, may focus excessively on features.

Results Summary for All Elicitation Methods

ARM seemed better suited to elicitation of security requirements than either IBIS or JAD. JAD seemed more suited to end-user functional requirements and provided no specific way to discuss quality requirements such as security. We found that IBIS was effective for documenting complex decision-making discussions but did not provide a structured way of generating security requirements.

We later experimented with other prioritization methods [1], notably Analytic Hierarchy Process (AHP), which seemed to provide more systematic prioritization than ARM.

Future Plans

These case studies are part of the security quality requirements engineering (SQUARE) project [13]. Current plans call for a comparison of SQUARE with other security requirements engineering methods, experimental combination of elicitation and prioritization methods (for example, combining ARM for elicitation with AHP for prioritization), development of supporting tools, and development of tutorial materials.◆

Acknowledgement

This work was part of a more extensive case study project by CMU graduate students Lydia Chung, Frank Hung, Eric Hough, and Don Ojoko-Adams [1]. This work is supported by the Army Research Office through grant number DAAD19-02-1-0389 ("Perpetually Available and Secure Information Systems") to CMU's CyLab.

References

1. Chung, L., et al. Security Quality Requirements Engineering (SQUARE): Case Study Phase III. Pittsburgh, PA: SEI, CMU, 2006 <www.sei.cmu.edu/publications/documents/06.reports/06sr003.html>.
2. Sindre, G., and A.L. Opdahl. "Eliciting Security Requirements by Misuse Cases." 37th International Conference on Technology of Object-Oriented Languages (Tools 37-Pacific 2000). Sydney, Australia, Nov. 20-23, 2000. Los Alamitos, CA: Institute of Electrical and Electronics Engineers Computer Society, 2000.
3. Checkland, P. Soft Systems Methodology in Action. Ontario, Canada: John Wiley & Sons, 1990.
4. QFD Institute. "Frequently Asked Questions About QFD." 2005 <www.qfdi.org/what_is_qfd/faqs_about_qfd.htm>.
5. Christel, M., and K. Kang. Issues in Requirements Elicitation. Pittsburgh, PA: SEI, CMU, 1992 <www.sei.cmu.edu/publications/documents/92.reports/92.tr.012.html>.
6. Systems Designers Scientific. CORE – The Method: User Manual. London, England: SD-Scicon, 1986.
7. Kunz, W., and H. Rittel. Issues as Elements of Information Systems. Berkeley: Institute of Urban and Regional Development, University of California, 1970 <www.iurd.ced.berkeley.edu/pub/WP-131.pdf>.
8. Wood, J., and D. Silver. Joint Application Development. 2nd ed. New York: John Wiley & Sons, 1995.
9. Kang, K.C., et al. Feature-Oriented Domain Analysis Feasibility Study. Pittsburgh, PA: SEI, CMU, 1990 <www.sei.cmu.edu/publications/documents/90.reports/90.tr.021.html>.
10. Schiffrin, D. Approaches to Discourse. Oxford, England: Blackwell Publishers Ltd, 1994.
11. Hubbard, R., Nancy R. Mead, and C. Schroeder. "An Assessment of the Relative Efficiency of a Facilitator-Driven Requirements Collection Process With Respect to the Conventional Interview Method." Proc. of the International Conference on Requirements Engineering. Los Alamitos, CA: IEEE Computer Society Press, 2000.
12. United States. Dept. of Homeland Security. BuildSecurityIn Portal. National Cyber Security Division <https://buildsecurityin.us-cert.gov/>.
13. Mead, N.R., et al. Security Quality Requirements Engineering (SQUARE) Methodology. Pittsburgh, PA: SEI, CMU, 2005 <www.sei.cmu.edu/publications/documents/05.reports/05tr009.html>.

About the Author

Nancy R. Mead, Ph.D., is a senior member of the technical staff in the Networked Systems Survivability Program at the SEI and is also a faculty member in the Master of Software Engineering and Master of Information Systems Management programs at CMU. Her research interests are in the areas of information security, software requirements engineering, and software architectures. Mead has more than 100 publications and invited presentations. She has a doctorate in mathematics from the Polytechnic Institute of New York and bachelor and master degrees in mathematics from New York University.

SEI
4500 Fifth AVE
Pittsburgh, PA 15213
E-mail: nrm@sei.cmu.edu

Requirements as Enablers for Software Assurance

Dr. Seok-Won Lee and Robin A. Gandhi
The University of North Carolina at Charlotte

The level of compliance with Certification and Accreditation (C&A) requirements conveys the level of assurance that one can expect from the quality of software behavior. Therefore, it is critical to understand the C&A requirements in terms of their applicability, scope, and impact of non-compliance on diverse aspects of software behavior to justify its certified qualities. However, numerous C&A requirements in ambiguous natural language descriptions with different levels of granularity are scattered across multiple documents that are used as guidance for C&A activities. As a result, a great deal of subjectivity is involved in understanding C&A requirements and their evaluation. This article discusses our approach to represent, model, and analyze C&A requirements to promote a common understanding among stakeholders for engineering more reliable software.

The Department of Defense (DoD) increasingly relies on the Defense Information Infrastructure (DII) that connects mission support, command and control, and intelligence computers and users through voice, data, imagery, video, and multimedia services, and provides information processing and other value-added services [1]. These services are dependent on the quality of underlying software, systems, practice, and environment to promote trust in the information furnished to the DoD and national-level decision makers. Therefore, the infrastructure-wide DoD Information Technology Security Certification and Accreditation Process (DITSCAP) [1] was introduced to ensure that the DoD's needs for software assurance are uniformly considered and maintained throughout the life cycle of all information systems that support information processing services within the DII.

The DITSCAP[1] is the standard DoD process for identifying information security requirements, providing security solutions and managing information systems security activities [1]. DITSCAP defines certification as the comprehensive evaluation of the technical and non-technical security features of an information system to establish the extent to which a particular design and implementation meets a set of specified security requirements. After this evaluation, the accreditation statement is an approval to operate the information system in a particular security mode using a prescribed set of safeguards at an acceptable level of risk. Using the DITSCAP to certify an information system is not simply a one-time process; it is maintained throughout the life cycle of the information system.

DITSCAP Limitations

Security requirements for DITSCAP certification address software assurance needs from diverse dimensions of process, organization, cost, time, data sensitivity, user clearance, system capabilities, development, deployment, maintenance, architecture, inventory, impact of non-availability, operational facilities, and other socio-technical aspects. Despite such a comprehensive coverage of software assurance needs, current DITSCAP practices have several limitations.

Practicing DITSCAP requires familiarity with several guidance documents from different levels in the DoD organizational hierarchy to identify the applicable set of security requirements necessary for certification. These documents include the DITSCAP application manual [2]; federal laws from the Office of Management and Budget; public laws; DoD and Department of Navy (DoN)[2] information assurance policies and implementations; National Institute of Standards and Technology (NIST) best practices for computer security; and many others. Each document usually ranges between 25 and 200 pages with heavy cross-referencing to other documents, making it extremely difficult to manually comprehend the interdependencies among their contents. In essence, the certification requirements with different levels of granularity in their specifications are scattered across multiple documents that provide guidance for C&A activities. These factors introduce a great deal of subjectivity in making decisions about the applicability, scope, and impact of non-compliance of DITSCAP security requirements. Table 1 summarizes these key decision points. Addressing these decision points with objective, justifiable, and repeatable criteria is critical to establish the assurance of reliable behavior of an information system subject to DITSCAP certification requirements. However, the interdependencies that exist among numerous certification requirements from multiple sources/documents severely complicate these decision points.

Due to the non-functional nature of DITSCAP certification requirements, these requirements often constrain diverse aspects of information system behavior in complex ways that are not readily apparent from their natural language descriptions. Additionally, understanding the interactions among various aspects of information system behavior is essential to reveal the cascading effects of the impact from the non-compliance of a certain certification requirement on overall system dependability.

Due to the limitation of current practices in addressing these issues, justifying compliance with certification requirements often satisfies a mere bureaucratic necessity without thoroughly understanding their consequences on the overall information system dependability and associated security risks. As a result, despite enormous efforts and resources currently spent on DoD software assurance initiatives, their effectiveness is only limited [3].

The Need for a Common Understanding of Requirements

Natural language security requirements for DITSCAP certification have little or no structural regularity in their specifications. Based on the seven facets of *complete* requirements – *who*, *where*, *what*, *when*, *why*, *which,* and *how* – a security requirement typically requires one to identify concepts related to 1) the assets that it protects, 2) the threats that it is driven by, 3) the vulnerabilities that it prevents, 4) the countermeasures that it suggests, 5) the mission criticality that it is subject to, 6) its source, 7) the goal of the security requirement, 8) the related stakeholders, and 9) other domain-specific concepts that need to be considered for creating a context that facilitates their uniform interpretation. However, most DITSCAP-enforced security requirements either do

Decision Point Categories	Key DITSCAP Decision Points
Applicability	DP1. Which regulatory documents should be used to identify C&A requirements?
	DP2. At what level of granularity should C&A requirements be identified?
	DP3. What are the types of the systems (for example, a major application or general support system) addressed by C&A requirements?
	DP4. What redundancies exist among C&A requirements and how should they be discovered?
Scope	DP5. Is the identified set of applicable C&A requirements complete?
	DP6. Who is responsible for or affected by (stakeholders) the C&A requirements?
Impact of Non-compliance	DP7. What are the criteria to assess requirements compliance?
	DP8. Do the compliance criteria provide a complete coverage of the different dimensions addressed by a given requirements?
	DP9. What are the risks associated with the system at a particular compliance level with C&A requirements?

Table 1: *DITSCAP Decision Points*

not explicitly identify these concepts or they are dispersed across multiple documents, making it difficult to practice DITSCAP and utilize its results for promoting software assurance.

To address these issues, we focus our research efforts on understanding and modeling the DITSCAP security requirements and related concepts in ways that support software assurance efforts. It is important to recognize that the assurance of trustworthy software behavior is determined by the needs of the problem domain. For example, in the DoD problem domain, security is of primary concern, whereas in the aviation problem domain, safety is of primary concern. Therefore, to effectively understand software assurance needs in the DoD as scoped by DITSCAP security requirements, we have produced a DITSCAP problem domain ontology. The meaning of ontology as adopted from the field of knowledge engineering refers to a set of concepts or terms that can be used to describe some area of knowledge or build a representation of it.

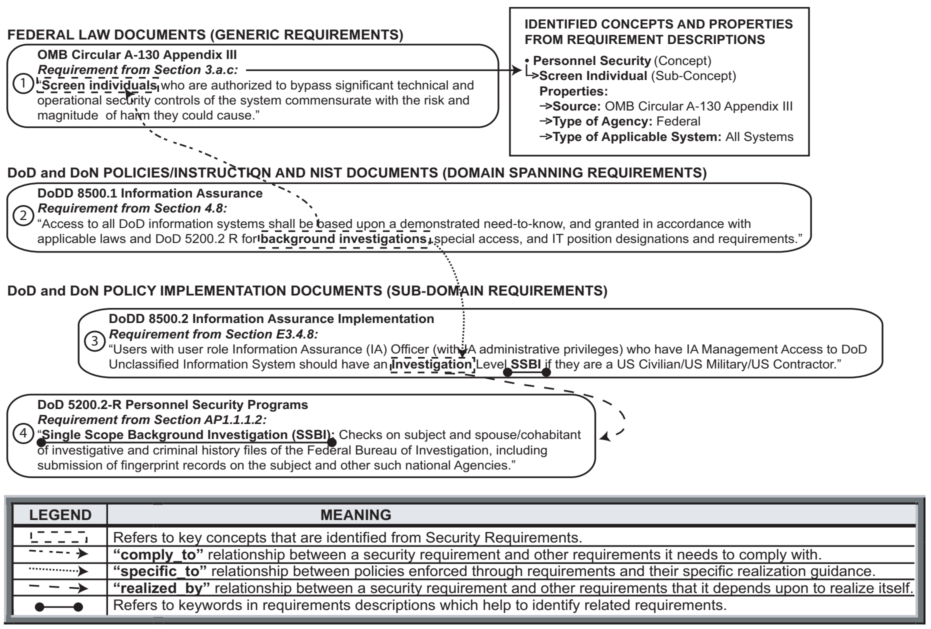
By combining techniques from requirements engineering and knowledge engineering we produce hierarchical ontological models [4] that characterize software assurance needs of the DoD. Specifically, we analyze each DITSCAP-related guidance document to extract ontological concepts that help in classifying and categorizing the DITSCAP security requirements from diverse dimensions. These ontological concepts are modeled as a hierarchy with several non-taxonomic interdependencies identified from DITSCAP-related guidance documents. The resulting ontology explicitly captures the concepts related to certification requirements and the relationships among them at different levels of granularity, previously scattered across multiple documents. The ontology promotes a common understanding among various stakeholders regarding DITSCAP security requirements specified within the federal, DoD, DoN, and NIST guidance documents at different levels of abstraction. Such a structured representation of DITSCAP security requirements facilitates a uniform understanding of their applicability, scope, and impact of non-compliance through its explicit traceable rationales and visual exploration capabilities.

Our approach to ontology development is primarily problem driven; its creation is guided based on the problem solving notions of goals, scenarios, and viewpoints (requirements engineering techniques) that effectively characterize the required dependable software behavior from diverse dimensions. The resulting integrated ontology is a human and machine understandable, hierarchical model of software assurance needs in the DoD, engineered using object-oriented ontological domain modeling techniques [5]. Goal-, scenario-, and viewpoint-based requirements engineering techniques are used to drive the identification of concepts related to a security requirement from DITSCAP-related guidance documents as structured representations of the following: 1) A hierarchy of requirement types that categorize security requirements from DITSCAP-related guidance documents; 2) A viewpoints hierarchy that models different perspectives and related stakeholders of a security requirement; 3) A risk assessment taxonomy that models risk factors from a broad spectrum of perceived risk sources identified in DITSCAP security requirements; 4) A hierarchy of C&A goals and related scenarios that models the DITSCAP process activities for gathering user/system criteria related to determining the applicability of security requirements; 5) A network-based information discovery taxonomy that aggregates results from network monitoring tools and scripts to assess compliance with security requirements in the actual environment; and 6) Interdependencies among various concepts in the DITSCAP ontology. These ontological concepts classify and categorize the certification requirements from multiple dimensions. Here, we briefly discuss the goal-driven process of identifying interdependent certification requirements categories scattered across multiple documents to produce a requirements hierarchy; however, further details about other models are in [5, 6] or available by contacting the authors.

Requirements Extraction and Modeling

As a first step towards understanding DoD software assurance needs based on DITSCAP security requirements, we identify the interdependencies among DITSCAP-related guidance documents using the cross-referential nature of their contents. We determine a hierarchical relationship among the generic federal-level documents, domain-spanning DoD and DoN policy/NIST security guidance documents, and site/agency specific DoD and DoN information assurance implementation guidance documents based on the level of abstraction pertaining to DITSCAP certification requirements specified within them.

Once the document interdependencies become apparent, a top-down goal decomposition approach systematically identifies interdependent requirements categories from multiple documents at different levels of abstraction. High-level assurance goals identified from certification requirements in federal-level guidance documents drive the elicitation of specific certification requirements which satisfy their parent goals from DoD/DoN/NIST guidance documents. Figure 1 (see page 22) elaborates on this process for the high-level assurance goal of *Screening Individuals* identified from certification requirements (annotated using the Label



FEDERAL LAW DOCUMENTS (GENERIC REQUIREMENTS)

1 OMB Circular A-130 Appendix III
Requirement from Section 3.a.c:
"**Screen individuals** who are authorized to bypass significant technical and operational security controls of the system commensurate with the risk and magnitude of harm they could cause."

IDENTIFIED CONCEPTS AND PROPERTIES FROM REQUIREMENT DESCRIPTIONS
- **Personnel Security** (Concept)
  - **Screen Individual** (Sub-Concept)
    Properties:
    → **Source:** OMB Circular A-130 Appendix III
    → **Type of Agency:** Federal
    → **Type of Applicable System:** All Systems

DoD and DoN POLICIES/INSTRUCTION AND NIST DOCUMENTS (DOMAIN SPANNING REQUIREMENTS)

2 DoDD 8500.1 Information Assurance
Requirement from Section 4.8:
"Access to all DoD information systems shall be based upon a demonstrated need-to-know, and granted in accordance with applicable laws and DoD 5200.2 R for **background investigations**, special access, and IT position designations and requirements."

DoD and DoN POLICY IMPLEMENTATION DOCUMENTS (SUB-DOMAIN REQUIREMENTS)

3 DoDD 8500.2 Information Assurance Implementation
Requirement from Section E3.4.8:
"Users with user role Information Assurance (IA) Officer (with IA administrative privileges) who have IA Management Access to DoD Unclassified Information System should have an **Investigation** Level **SSBI** if they are a US Civilian/US Military/US Contractor."

4 DoD 5200.2-R Personnel Security Programs
Requirement from Section AP1.1.1.2:
"**Single Scope Background Investigation (SSBI)**: Checks on subject and spouse/cohabitant of investigative and criminal history files of the Federal Bureau of Investigation, including submission of fingerprint records on the subject and other such national Agencies."

LEGEND	MEANING
⌐- - -¬	Refers to key concepts that are identified from Security Requirements.
- - - →	**"comply_to"** relationship between a security requirement and other requirements it needs to comply with.
········→	**"specific_to"** relationship between policies enforced through requirements and their specific realization guidance.
— — →	**"realized_by"** relationship between a security requirement and other requirements that it depends upon to realize itself.
●—●	Refers to keywords in requirements descriptions which help to identify related requirements.

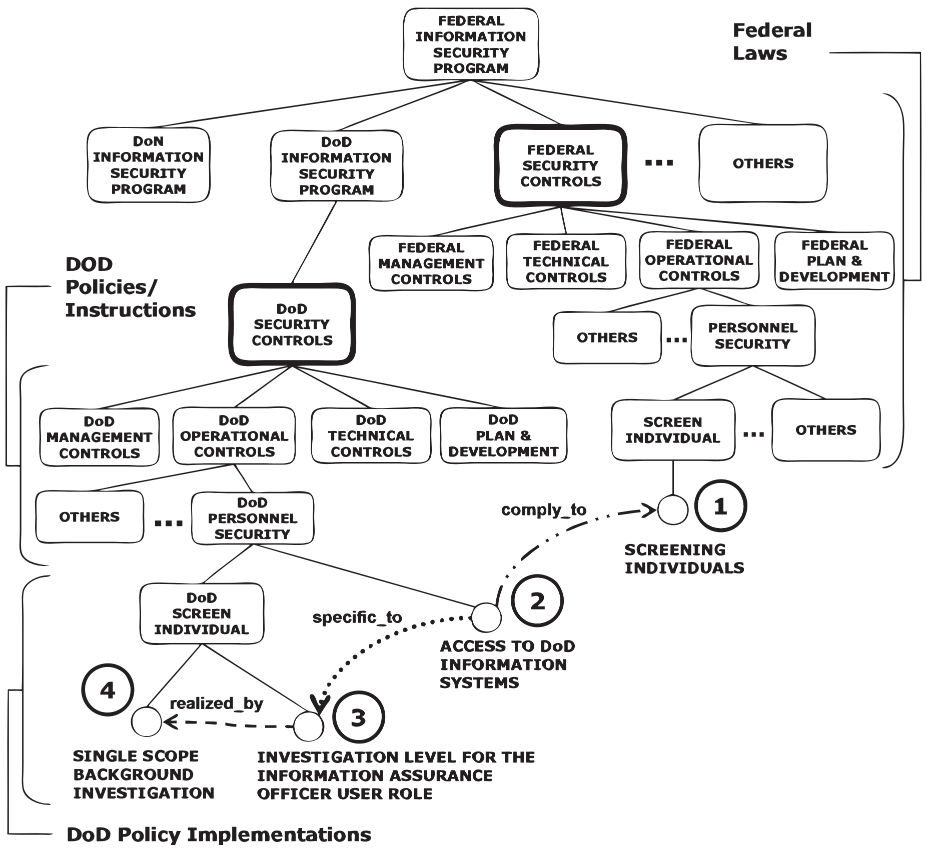
Figure 1: *Extraction of Security Requirements, Categories, Properties, and Their Interdependencies From DITSCAP-Related Guidance Documents*

1) expressed in a federal-level document. To satisfy this goal, we identify specific certification requirements pertaining to *background investigation* and other relevant security requirements categories (annotated using the Labels 2, 3, and 4 in Figure 1) from DoD/DoN/NIST documents. This process explicitly models the security requirement categories, their properties, and the relationships among them, spanning multiple DITSCAP-related guidance documents from different levels in the DoD organizational hierarchy.

The requirements hierarchy, modeled within the DITSCAP ontology, aggregates these requirements categories through a hierarchical representation that includes top-level generic requirements categories, mid-level domain spanning requirements and low-level agency specific requirements. Such a hierarchical organization of requirements types allows for a systematic exploration of DITSCAP security requirements during certification activities. A partial requirements hierarchy that explicitly models the requirements categories and their relationships identified in Figure 1 is depicted in Figure 2 with corresponding requirement labels. The security requirements hierarchy also promotes consistency in managing requirements from multiple documents by providing a generic set of categories. For example in Figure 2, the *Federal Security Controls* and *DoD Security Controls* requirements categories provide consistency and traceability among certification requirements extracted from federal and DoD C&A guidance documents respectively.

Our ontology development efforts are supported by the GENeric Object Model (GenOM) toolkit [7]. GenOM is an integrated development environment (developed at the University of North Carolina - Charlotte [UNCC] and available for use) for ontological engineering processes with functionalities to create, browse, access, query, and visualize associated ontologies. GenOM is compatible with the Web Ontology Language representation [8] and is associated with an inference engine that supports reasoning upon the ontological concepts and relationships modeled in its knowledge bases.

Figure 2: *A Partial Requirements Hierarchy*


FEDERAL INFORMATION SECURITY PROGRAM
Federal Laws
DoN INFORMATION SECURITY PROGRAM
DoD INFORMATION SECURITY PROGRAM
FEDERAL SECURITY CONTROLS
...
OTHERS
FEDERAL MANAGEMENT CONTROLS
FEDERAL TECHNICAL CONTROLS
FEDERAL OPERATIONAL CONTROLS
FEDERAL PLAN & DEVELOPMENT
DOD Policies/ Instructions
DoD SECURITY CONTROLS
OTHERS
...
PERSONNEL SECURITY
DoD MANAGEMENT CONTROLS
DoD OPERATIONAL CONTROLS
DoD TECHNICAL CONTROLS
DoD PLAN & DEVELOPMENT
SCREEN INDIVIDUAL
...
OTHERS
OTHERS
...
DoD PERSONNEL SECURITY
comply_to
1
SCREENING INDIVIDUALS
DoD SCREEN INDIVIDUAL
specific_to
2
ACCESS TO DoD INFORMATION SYSTEMS
4
realized_by
3
SINGLE SCOPE BACKGROUND INVESTIGATION
INVESTIGATION LEVEL FOR THE INFORMATION ASSURANCE OFFICER USER ROLE
DoD Policy Implementations

Metrics and Measures for Compliance With Certification Requirements

For each certification requirement modeled within the requirements hierarchy, the DITSCAP ontology development also involves the creation of compliance questionnaires (representative of various compliance metrics) with well-defined answer options (representative of compliance measures). These questionnaires systematically gather evidences for the target information system to determine its eligibility for DITSCAP certification. Utilizing DITSCAP-related guidance documents and domain expertise, we establish the criteria addressed by questionnaires. Responses to these questionnaires are gathered by consulting various sources related to the target information system such as users, operating manuals, plans, architecture diagrams, or through network monitoring tools and scripts. These responses establish the extent to which the higher-level requirements categories in the requirements hierarchy are satisfied through specific policies, procedures, or technical rationales in the actual environment. The questionnaires introduce uniformity in the evaluation of security requirements while avoiding subjective interpretations of certification requirements compliance criteria.

The compliance information gathered for DITSCAP certification requirements can also be interpreted in terms of other models within the DITSCAP ontology. The natural language descriptions of DITSCAP security requirements embody concepts which help in establishing these relationships. From requirements descriptions we identify concepts related to stakeholders in the viewpoints hierarchy; C&A process goals in the goal hierarchy; risk factors of threat, vulnerabilities, countermeasures, assets, and mission criticality in the risk assessment taxonomy; and actual system characteristics captured through the network-based information discovery taxonomy. Such relationships for the DITSCAP certification requirement of *Enclave Boundary Defense* with other concepts within the DITSCAP ontology are visualized in Figure 3. Such explicitly modeled relationships have also helped in perceiving the operational risks based on the level of compliance of the target information system with DITSCAP security requirements [9].

Benefits of Our Approach

Our approach for DITSCAP security requirements modeling and analysis has many of the following potential benefits:

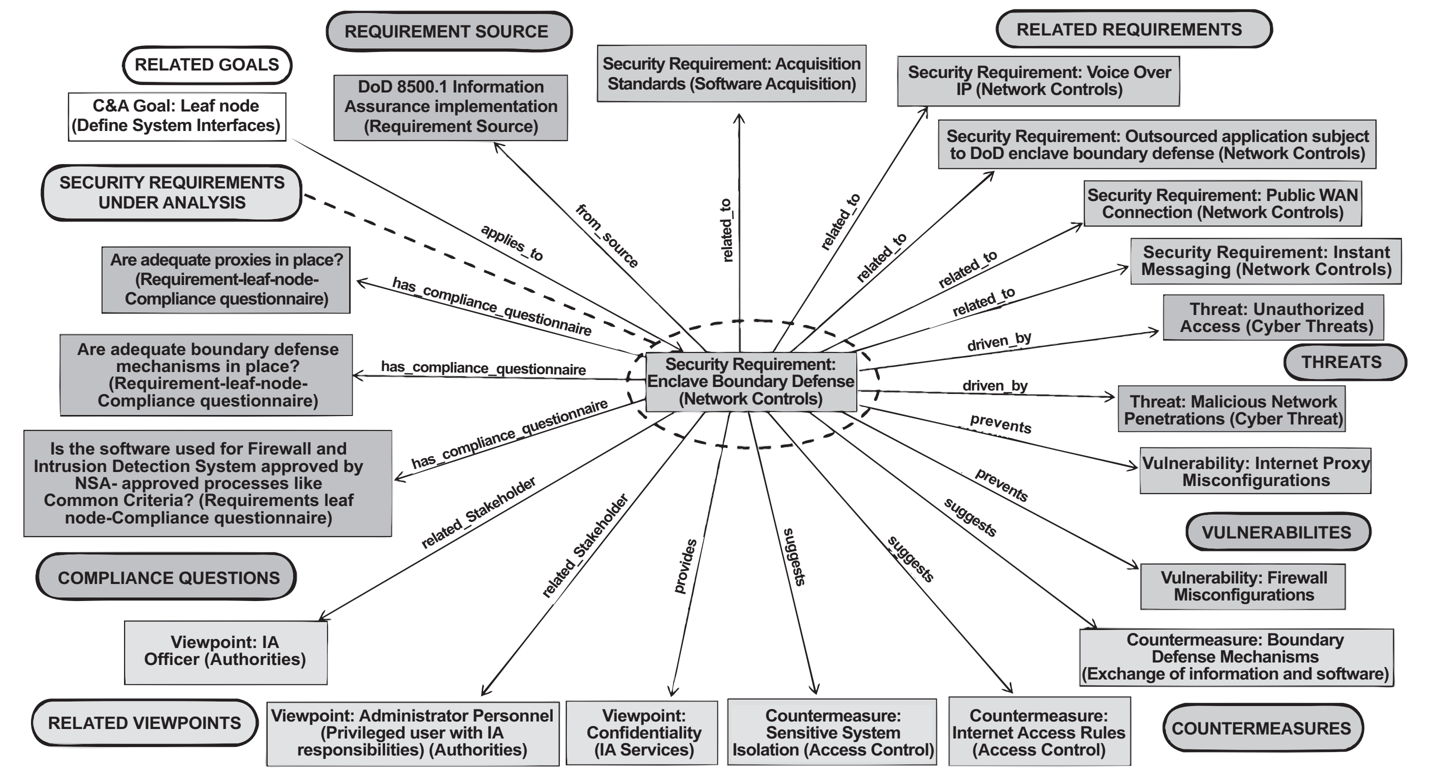
- A reusable and configurable repository of certification requirements, policies, and directives. This gives users the ability to map and reflect the appropriate language of existing requirements applicable to their agency.
- Intuitive Graphical User Interfaces can be supported through the DITSCAP ontology to guide C&A process activities.
- Currently, no systematic methods exist to collect information related to the compliance level of certification requirements. Additionally, a long and exhaustive task of gathering requirement compliance criteria from the target system results in a subjective and ad-hoc C&A process. To address these issues, an ontology-driven methodology to gather compliance information using well-defined questionnaires (metrics and measures) for certification requirements provides objective and uniform criteria to facilitate cost-effective decision making.
- The information gathered about the target information system through the requirements compliance questionnaires can be transformed into the required form of documentation for reuse across multiple software assurance initiatives, saving costly rework.
- The hierarchical representation of DITSCAP ontology provides the flexibility of communicating compliance results at different levels in the organization and sharing them among agencies based on a common understanding.
- Reducing the certification costs due to the need of fewer resources to conduct, manage, and maintain C&A activities. Efficient C&A activities can significantly reduce the development and deployment time of more reliable information systems.

Current Status, Challenges and Next Steps

Through our efforts, a prototype DITSCAP automation tool has been developed. We are currently in the process of outlining a case study designed research methodology where a group of experts perform C&A activities with and without using the DITSCAP ontology and related tool support. The results of this case study with evaluation metrics and measures will eventually serve as a basis for establishing the benefits of our approach.

We realize that based on the given target information system, it is essential to discover *links* among non-functional certification requirements which may originate from different dimensions, but are necessary to collectively ascertain overall reliable emergent software behavior. Although such links cannot be anticipated or formalized for all situations, we will explore ways to utilize the traceability offered within the DITSCAP ontology to help experts gradually hypothesize more meaningful relationships among

Figure 3: *Visualization of a DITSCAP Security Requirement and Its Relationships With Other Concepts*


REQUIREMENT SOURCE
RELATED REQUIREMENTS
RELATED GOALS
Security Requirement: Acquisition Standards (Software Acquisition)
Security Requirement: Voice Over IP (Network Controls)
DoD 8500.1 Information Assurance implementation (Requirement Source)
C&A Goal: Leaf node (Define System Interfaces)
Security Requirement: Outsourced application subject to DoD enclave boundary defense (Network Controls)
SECURITY REQUIREMENTS UNDER ANALYSIS
Security Requirement: Public WAN Connection (Network Controls)
applies_to
from_source
related_to
related_to
related_to
related_to
related_to
Security Requirement: Instant Messaging (Network Controls)
Are adequate proxies in place? (Requirement-leaf-node-Compliance questionnaire)
has_compliance_questionnaire
Threat: Unauthorized Access (Cyber Threats)
driven_by
THREATS
Are adequate boundary defense mechanisms in place? (Requirement-leaf-node-Compliance questionnaire)
has_compliance_questionnaire
Security Requirement: Enclave Boundary Defense (Network Controls)
driven_by
Threat: Malicious Network Penetrations (Cyber Threat)
prevents
has_compliance_questionnaire
Is the software used for Firewall and Intrusion Detection System approved by NSA- approved processes like Common Criteria? (Requirements leaf node-Compliance questionnaire)
Vulnerability: Internet Proxy Misconfigurations
prevents
related_Stakeholder
suggests
VULNERABILITES
related_Stakeholder
provides
suggests
suggests
COMPLIANCE QUESTIONS
Vulnerability: Firewall Misconfigurations
Viewpoint: IA Officer (Authorities)
Countermeasure: Boundary Defense Mechanisms (Exchange of information and software)
RELATED VIEWPOINTS
Viewpoint: Administrator Personnel (Privileged user with IA responsibilities) (Authorities)
Viewpoint: Confidentiality (IA Services)
Countermeasure: Sensitive System Isolation (Access Control)
Countermeasure: Internet Access Rules (Access Control)
COUNTERMEASURES

non-functional certification requirements while understanding their consequences on the overall dependable behavior of the target information system.

Due to the nature of the ontological engineering, currently the DITSCAP ontology has been constructed manually using frequent feedback and refinement from experts. We also explore techniques for automatically processing natural language guidance documents and identify ontological concepts that experts can refine further.

Reaching Out

In general, our approach helps in capturing the characteristics of information present sparsely in documents and the way these characteristics can be represented using ontological modeling processes to infer valuable knowledge that assists decision making activities. Hence, we contend that our methodology can be favorably extended to non-DITSCAP uses (e.g. Federal Information Security Management Act, NIST, Common Criteria, or the Health Insurance Portability and Accountability Act) where the decision making activities require sifting through large volumes of information.

Our research efforts seamlessly complement the migration from DITSCAP to DoD Information Assurance Certification and Accreditation Process (DIACAP) for future DoD endeavors of the Global Information Grid and net-centric dynamic C&A [10]. The DIACAP Enterprise Mission Assurance Support System that standardizes approaches for describing and collecting data for C&A can leverage the benefits of an ontological representation of security requirements to promote uniformity, reusability, and portability, as well as the sharing of results from C&A activities. The DITSCAP ontology also facilitates the interpretation of results from network monitoring tools and scripts in terms of their impact to compliance with certification requirements, providing interesting research directions for the DIACAP Vulnerability Assessment Management Service.◆

Acknowledgement

This work is partially supported by the grant from the Critical Infrastructure Protection Center, Space and Naval Warfare Systems Center, Charleston, South Carolina.

References

1. DoD. "DoD 5200.40. DITSCAP." Dec. 1997 <www.dtic.mil/whs/directives/corres/html/520040.htm>.
2. DoD. "DoD 8510.1-M, DITSCAP." Application Manual. 2000 <www.dtic.mil/whs/directives/corres/html/85101m.htm>.
3. Davis, T. "No Computer System Left Behind: A Review of the 2005 Federal Computer Security Scorecard." Press Release. Government Reform Committee, 2005 <http://reform.house.gov/UploadedFiles/TMDFISMA06Opener.pdf> .
4. Swartout, W. and A. Tate. "Ontologies." IEEE Intelligent Systems 141. (1999): 18-19.
5. Lee, S., D. Muthurajan, and R. A. Gandhi, et al. "Building Decision Support Problem Domain Ontology From Natural Language Requirements for Software Assurance." International Journal on Software Engineering and Knowledge Engineering (2006).
6. Lee, S.W., R.A. Gandhi, and Gail-Joon Ahn. "Certification Process Artifacts Defined as Measurable Units for Software Assurance." International Journal on Software Process: Improvement and Practice (2006).
7. Lee, S.W., and D. Yavagal. "GenOM User's Guide Vers. 2.0." Technical Report TR-NiSE-05-05. Knowledge Intensive Software Engineering Research Group. Dept. of Software and Information Systems: UNCC, 2005.
8. McGuinness, D., and F. van Harmelen, Eds. "OWL Web Ontology Language Overview." W3C Recommendation, 2004 <www.w3.org/TR/owl-features/>.
9. Lee, S.W., R.A. Gandhi, and G.J. Ahn. "Security Requirements Driven Risk Assessment for Critical Infrastructure Information Systems." Proc. of the Symposium on Requirements Engineering for Information Security (SREIS 05), 2005.
10. Turner, G., P. Holley, E.J. Mehan, and M. Colon. "Net-Centric Assured Information Sharing – Moving Security to the Edge Through Dynamic Certification and Accreditation." IA Newsletter 8.3 (Winter 2005/2006).

Notes

1. DITSCAP is currently undergoing a migration to the DoDI 8510.bb, DIACAP. However, it does not affect the utility of the approaches outlined in this article.
2. Although we address our approach in the context of the DON, our techniques are generally applicable to C&A standards for other agencies.

About the Authors

Seok-Won Lee, Ph.D., is an assistant professor of software and information systems at UNCC. Prior to UNCC, he was affiliated with Science Applications International Corporation and the IBM T.J. Watson Research Center. Lee's areas of specializations include software engineering and knowledge engineering with specific expertise in ontology-based requirements engineering, knowledge acquisition, and machine learning. Lee is currently focusing on new research in the areas of knowledge-intensive software engineering, software evaluation research, object-oriented domain modeling and their applications to information security and assurance. He holds a master's degree in computer science from the University of Pittsburgh and a doctorate in information technology from George Mason University.

Robin Gandhi is pursuing a doctorate in information technology and has been a research assistant in the Department of Software and Information Systems at UNCC since 2003. His research interests include requirements engineering, knowledge-intensive software engineering, and ontology-based object-oriented domain modeling and analysis. Gandhi received his undergraduate degree in electronics engineering from Sardar Patel University, India, and his Master of Science in computer science from UNCC.

Both authors can be reached at:
Department of Software and Information Systems
UNCC
9201 University City BLVD
Charlotte, NC 28223-0001
Phone: (704) 687-8662/8385
Fax: (704) 687-4893
E-mail: seoklee@uncc.edu, rgandhi@uncc.edu

Finding and Fixing Problems Early: A Perspective-Based Approach to Requirements and Design Inspections

Dr. Jeffrey C. Carver
Mississippi State University

Dr. Forrest Shull and Dr. Ioana Rus
Fraunhofer Center for Experimental Software Engineering

Viewing security vulnerabilities as a specific type of software defect allows proven software engineering techniques for finding and fixing them to be used early in the development of the product. Finding and fixing these problems early (i.e. at the requirements or design phase) will reduce the overall risk and cost of the product. This article describes the application of a previously successful early life cycle software inspection approach (perspective-based reading [PBR]*) to the problem of software security. Excerpts from this tailored approach are provided along with guidance on it use.*

Developing secure software requires engineering and management attention, as well as extra effort and resources. Rather than being treated as an *add-on*, security must be built in throughout the software life cycle, using a mixture of good software engineering practices (to ensure quality software in general) and good security practices (to ensure that exploitable vulnerabilities are not present). Integrating security practices with software engineering practices throughout the development process helps an organization anticipate security failures and develop software that can sustain attacks.

Currently, the predominant method for finding vulnerabilities is to *penetrate and patch*. This approach focuses most of the effort on the latter stages of the software life cycle, e.g., testing for a set of known vulnerabilities or waiting for a vulnerability to be exploited in delivered code. Only after such a vulnerability is found does the development team modify the software to address that vulnerability [1]. The effectiveness of this approach is highly correlated with the quality of the testing activities. An experienced tester can be quite effective, but an inexperienced tester can miss many important issues.

As evidence of the problems associated with testing for security vulnerabilities, although sensitivity to and focus on security problems has increased in recent years, the number of software vulnerabilities found in deployed software has actually increased – not decreased. For example, according to the United States Computer Emergency Readiness Team (US-CERT), the number of new vulnerabilities discovered in software has been growing at rates close to 140 percent, recently in excess of 5,000 per year. Other vulnerability collections show similar trends – e.g., the National Vulnerability Database has nearly 13,000 documented vulnerabilities, and Bugtraq, a moderated mailing list where vulnerabilities are reported and discussed, holds discussions on about 400 items per month. These figures make it clear that there is a national need to better address software security.

An improvement over this *penetrate and patch* mentality is to *build security in* from the beginning of the product lifecycle [2, 3]. Governmental organizations like the Department of Homeland Security (DHS) have already realized the benefits of this approach and are promoting research to make it happen. Standard software engineering approaches for building in software quality provide some insight into building in security. Traditionally, software engineering has defined a *defect* as an error, fault, or failure in the software system or its related artifacts. Security vulnerabilities are a special type of these more general software engineering defects and therefore benefit from similar detection and removal approaches.

Although no concrete guidelines exist, there is a growing recognition that early life-cycle issues do play an important role in the development of secure software. Problems in the early life-cycle phases have caused some costly and highly visible problems leading to untrustable systems. A well-publicized example of this type of problem was the theft of personal data from a database of background files on most American citizens maintained by the ChoicePoint corporation. The theft of data occurred when criminals set up fake companies (e.g. debt collectors, insurance agencies) and gained access to ChoicePoint's databases [4]. This security vulnerability can be viewed as a requirements and design problem. Had the creators of the software included requirements for verifying the legitimacy of a company prior to allowing access to private information, this theft could have likely been prevented.

Security experts have also clearly recommended the need for early life cycle work to address security vulnerabilities [5]. All of this evidence combines to create a powerful message: Although often ignored, decisions made (or missed) in the early life cycle have a large impact on the level of security achievable in the final system.

This message is quite familiar to software engineers. Software engineering practitioners and researchers have observed (and measured) that the earlier defects are found in the software life cycle the easier and cheaper they are to repair [6]. Many software engineering best practices are concerned with how to apply early life-cycle verification and validation (V&V) practices to *build quality in* throughout the life cycle rather than *test it in* during late life-cycle testing phases. Viewing security vulnerabilities as a special type of defect allows for building security into systems in the same way as building quality into systems. The ultimate goal is to integrate the best practices from the security engineering and software engineering communities into a set of techniques for identifying and removing security vulnerabilities early in the software life cycle. This article illustrates the adaptation of one such technique, PBR, to address the security vulnerability problem during a requirements inspection process.

An Early Life-Cycle Approach to Security

A large number of software engineering studies have established inspections as an effective method for reducing defects in software systems [6]. An inspection is a static review process in which a software artifact (e.g. requirements document, design document, or code) is reviewed by one or more inspectors to verify that it meets a set of quality properties. Companies like Microsoft have recognized the potential reduction of vulnerabilities that inspections can cause in the early life-cycle phases and have created their own checklists focused on early life-cycle issues.

One of the most successful examples of inspection use came from the Software Engineering Laboratory (SEL) at NASA's Goddard Space Flight Center. Figure 1 (see page 26) illustrates the impressive reduction in defect rates (per thousand developed lines of code [DLOC]) achieved on software projects over a nineteen year period at the SEL. This chart shows sustained continuous improvement over a long period of time in an

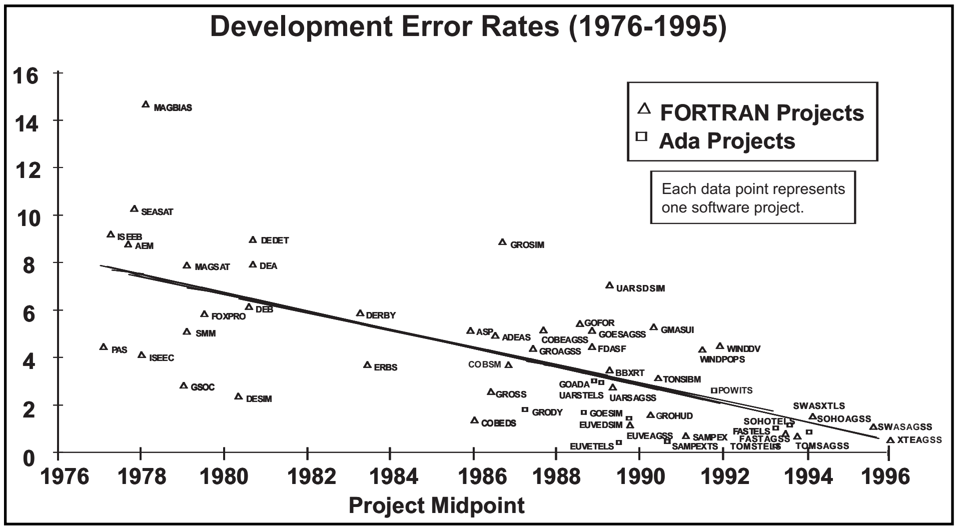


Figure 1: *Demonstrated Defect Reduction in Errors per 1,000 DLOC at NASA's SEL*

organization that built real-time software systems to support safety critical missions costing millions of dollars. Although much effort was spent on software quality improvement at Goddard during this time period, Frank McGarry and Mike Stark, two former directors of the SEL, have both credited the introduction and maintenance of inspections as being the most important factor in the error rate reduction.

One of the difficulties in performing early life-cycle reviews is the amount of human judgment required. Static analysis tools do exist, but these tools focus mainly on code. Early life-cycle V&V by its very nature is human intensive: Identifying and predicting the impacts of requirement or design decisions is not easily amenable to an automated checking approach. To be done effectively, these tasks require the experience and knowledge necessary to make good judgment calls. For security problems, this means that security expertise is essential, but often, lacking in software engineers. Therefore, a mechanism is needed to supplement the software engineers' knowledge with security specific knowledge. PBR is a technique that can be used for encoding and transferring security-relevant expertise.

A Tailorable, Perspective-Based Approach

Perspective-based inspection is a variant of a formal technical review, which provides a useful framework for integrating security concerns. This type of inspection is based on explicitly defining the important stakeholders for a particular artifact and the types of issues that are of importance to the team. Rather than asking each reviewer to search for all types of problems, the perspective-based approach requires inspectors to examine the document using a role-based *scenario* based on how one specific stakeholder would use the document to build the system. For example, an inspection of system requirements includes an inspector using a tester perspective. This inspector reviews the requirements by following a scenario in which he/she considers how to generate test cases based on the requirements. Any time the inspector experiences difficulty in using the document to complete his scenario he records this difficulty as an issue that must be fixed. This list of issues then becomes the list of items that is returned to the author of the document for repairs.

The set of appropriate stakeholders and issue types must be tailored for different environments. In this sense, the perspective-based approach provides a set of guidelines for creating an effective, tailored inspection technique, not a one-size-fits-all process.

By emphasizing the stakeholders, the perspective-based approach provides a helpful way to make the need for collaboration between software and security engineers explicit. Each of these roles must be represented by at least one inspector on the review team. This approach is also useful when it is not possible to get an inspector with sufficient security expertise to participate in the inspection process. Creating a scenario based on the way each stakeholder uses the document captures that stakeholder's expertise about early life-cycle indicators of potential security vulnerabilities that can be used by more novice inspectors.

To conduct a perspective-based inspection, practitioners first need to decide what documents should be inspected. Inspection of software-specific documents (like the requirements specification or design document) needs to be augmented with the inspection of security-specific artifacts (such as threat models). The goal of the inspection must be to ensure that the security models are internally consistent and correct, as well as that their implications for the system are adequately reflected in the software work products, to support construction of a correct end-product.

For the documents that have been selected, a list of stakeholders must be compiled, considering the following:

- *Stakeholders in downstream phases* who need the document to perform their own job. Related to security, the following are some examples: testers, who need to ensure that security requirements are clearly stated in terms of their expected behavior and the functionalities to which they should be applied; designers and/or developers, who need to ensure that security policies are specified in enough detail to allow for correct implementation; and users, who need to ensure that the final behavior of the system, including all security policies, will meet their needs.
- *Stakeholders from previous phases* who want to ensure that their decisions relevant to system design are adequately reflected in the document. Related to security, an important stakeholder is the developer of threat models and other early life-cycle security artifacts, who needs to ensure that the behavior identified by these models is in the system artifacts.
- *Stakeholder's specific types of expertise* need to be correctly reflected in the software work documents. An important security-related perspective is a *black hat* user who focuses specifically on issues that could lead to exploitable security vulnerabilities and increases the risk of a successful attack on the software.

For each perspective identified, a scenario that reflects the normal day-to-day work activity of the respective stakeholder must be created. Different stakeholders' scenarios require the inspectors to focus on different aspects of the document, while the entire set of perspectives covers the whole artifact. By focusing each reviewer on a separate task, the overlap among the responsibility of various inspectors is reduced. At the same time, the importance and necessity of the role played by each inspection member increases because no two reviewers focus on the same set of defects. This approach combats the assumption that simply having more eyes on a document increases the chances of finding problems. In reality, we find the opposite to be true: Having more people look at the same document without a specific focus allows each inspector to assume that his/her expertise and time are not crucial, and that someone else will find any problems they miss. This is a potentially dangerous assumption.

Our previous experiences in organizations such as NASA have shown perspective-based inspections to be especially useful for helping bring novice reviewers up to speed, by giving them a clear direction of *how* to get started on their analysis of the document, and by giving them a clear subset of concerns to focus on rather than having to *wear all the hats*. Perspective-based inspections were evaluated at Goddard with a controlled experiment comparing different approaches for reviewing requirements specifications. Using the perspective-based approach allowed individual reviewers to find up to 30 percent more defects, on average, than when they used the standard NASA review approach. When team results were statistically simulated from the individual data, teams using perspectives also found up to 30 percent more defects than teams using the standard NASA approach. These differences were statistically significant [7].

This study has been replicated multiple times with similar results. A series of other studies provide support for the benefits of using a perspective-based approach for inspections of different types of artifacts and different organizations:

- A study conducted at Lucent Technologies indicated that inspection teams using perspectives to find defects in formal requirements models were significantly more effective than teams using only checklists or unstructured techniques [8].
- A study at a U.S. government organization in 1998 showed that teams found about 30 percent more usability problems in Web interfaces when using a perspective-based inspection approach [9].
- A study of German software professionals from various companies producing object-oriented designs showed that teams using the perspective-based approach found 41 percent more defects than teams using only checklists, and the cost per defect was significantly lower [10].
- A study of code inspections at Bosch Telecom in Germany indicated that teams using perspectives were more effective at finding defects than teams using the baseline inspection approach with the *improvement* being statistically significant in two of the three experimental runs. The cost per detected defect was also significantly lower for the perspective-based approach in all runs [11, 12].
- A study of object-oriented design inspections at Ericsson in Sweden showed that teams using perspectives found more defects than those using their usual approach, although the perspective-based inspections took more time [12].

Tailoring Perspective-Based Reading for Security

Based on the previous proven success of using the perspective-based approach to find generic software quality defects, we have tailored this approach to focus on software security. In this paper, we describe a set of perspectives for a requirements inspection. To perform this tailoring, we augmented two of the *standard* PBR perspectives (the designer and the tester) with additional security specific questions. In addition, we created a new perspective based on the needs of a *black hat* tester. The remainder of this section briefly describes each technique.

The *designer* perspective has the goal of ensuring that there is enough, consistent information present in the requirements to successfully create a system design. The existing scenario is augmented with questions that focus on whether important security-related information has been correctly specified rather than being left up to the designer, who may not be familiar with all details of the security policy. Examples of the new questions that the reviewer using the designer perspective should consider when following this perspective include the following:

- Have the requirements specified enough information about the security policies for the designer to understand whether a layered security policy is required instead of a single point of vulnerability?
- If several administrator roles are defined, have they been defined as separate accounts with limited access to security resources, or a single account with comprehensive *super user* permissions?

In a similar fashion, the scenario for the *tester* perspective remains unchanged, but is augmented with security specific questions. The inspector using the tester perspective has the goal of ensuring that the trustworthiness of the system will be knowable during the testing phase. Examples of the new questions that the reviewer using the tester perspective should consider include the following:

- Have the requirements specified appropriate exception-handling functionality?
- Have the requirements specified adequate safeguards that would take effect once a malicious user has gained unauthorized access to the system?
- Does the system have a well-defined status, either a secure failure state or the start of a plausible recovery procedure, after a failure condition?

Finally, the *black hat* perspective is a new one (i.e. not tailored from the previous set of perspectives) which focuses the reviewer on finding weaknesses in the requirements that could be exploited via an attack. The scenario that the reviewer follows is to create a set of malicious attack scenarios that seek to exploit system vulnerabilities. While creating this scenario, the reviewer focuses on three types of information relevant at the requirements stage: Cryptography, Authentication/Authorization, and Data Validation. These types of information, along with the related questions were adapted for requirements from Araujo and Curphey's article on Security Code Reviews [13].

Cryptography relates to the encoding mechanisms specified for data items within the system. During the review, the inspector is looking for underspecified or incorrectly specified features that could be exploited. Example questions include the following:

- Can the encoding mechanisms specified for transmission and storage of data be broken?
- Do the cryptographic mechanisms specified follow well-known, well-documented, and publicly scrutinized algorithms, and if not, can they be easily broken?

Authentication/Authorization focuses the reviewer on determining how unauthorized users could gain access to the system. Example questions include the following:

- Can the protocols for validating user identity be broken?
- If account lockout is specified, are there requirements in place to prevent denial-of-service attacks?
- Can user privileges be artificially elevated due to omissions or poorly specified requirements?

Data Validation is an important source of security vulnerabilities and focuses the reviewer on determining whether invalid data could be entered into the system. An example question: Do the requirements leave any opportunities for invalid data to be entered by the lack of validation of external data?

These excerpts from the requirements review techniques illustrate that formulating inspection methods must fully leverage the intelligence and flexibility of the human beings involved in the procedure. As much as possible, inspectors should avoid using algorithmic heuristics that would be candidates for tool support instead (for example, keep coupling low by ensuring that no class calls more than six others). Rather, inspectors should be given tasks that require both semantic understanding of the system and judgment calls. The flexibility of such an approach allows inspectors to focus both on bad things that should be avoided (excessive coupling) and good things that are omitted (exception handling for appropriate functionalities). Similar ideas can be incorporated into the reviews of the threat models, architecture and design documents by adding similar questions specific to each artifact.

Conclusions

The decisions made in early life-cycle development phases have a large impact on whether secure systems are achievable or not. Just as for other types of quality issues, early life-cycle V&V activities can detect and repair security issues early on, for example, by checking that security requirements are well thought-out, feasible, and consistent with user needs; that system architectures exhibit good design principles, making them easier to maintain and fix without introducing vulnerabilities; or that components are designed so that if security measures are breached in one component, an attacker gains access only to a limited part of the system. Also, as for other types of quality issues, finding security problems early saves time and effort for the development team by avoiding the need to fix the many downstream documents that instantiate early decisions.

A perspective-based inspection, as illustrated in this article, is one approach which has been very effective at early life cycle defect detection and can be used to tailor V&V techniques to focus on security. While other approaches are certainly possible, the explicit reliance of the perspective-based approach on the needs of the stakeholders gives practitioners a way to bring the right expertise to bear and capture best practices so they can be employed by a larger set of inspectors.◆

References

1. McGraw, G. “Building Secure Software: Better Than Protecting Bad Software.” IEEE Software 19.6 (2002): 57-58.
2. Beaver, K. and Sima, C. “Software Development: Building Security In.” Security.itworld. 5 Sept. 2006.
3. Grance, T., Hash, J., and Stevens, M. “Security Considerations in the Information System Development Life Cycle.” NIST Special Publication 800-64, 2004.
4. Sullivan, B. “Data Theft Affects 145,000 Nationwide.” MSNBC. 18 Feb. 2005.
5. McGraw, G. Software Security: Building Security In. Addison-Wesley Professional, 2006
6. Shull, F., et al. “What We Have Learned About Fighting Defects.” Proceedings of IEEE Symposium on Software Metrics. 2002.
7. Basili, V., et al. “The Empirical Investigation of Perspective Based Reading.” Empirical Software Engineering – An International Journal 1.2 (1996): 133-164.
8. Porter, A. and Votta, L. “Comparing Detection Methods for Software Requirements Inspections: A Replication Using Professional Subjects.” Empirical Software Engineering – An International Journal 3.4 (1998): 355-379.
9. Zhang, Z., Basili, V., and Shneiderman, B. “Perspective-Based Usability Inspection: An Empirical Validation of Efficacy.” Empirical Software Engineering – An International Journal 4.1 (1999): 43-70.
10. Laitenberger, O., Atkinson, C., Schlich, M., and El Emam, K. “An Experimental Comparison of Reading Techniques for Defect Detection in UML Design Documents.” Journal of Systems and Software 53.2 (2000):183-204.
11. Laitenberger, O., El Emam, K., and Harbich, T.G. “An Internally Replicated Quasi-Experimental Comparison of Checklist and Perspective Based Reading of Code Documents.” IEEE Transactions on Software Engineering 27.5 (2001): 387-421.
12. Conradi, R., et al. “Object-Oriented Reading Techniques for Inspection of UML Models – An Industrial Experiment.” Proceedings of European Conference on Object-Oriented Programming (ECOOP ’03), Darmstadt, Germany, 2003.

13 Araujo, R. and Curphey, M. “Software Security Code Review: Code Inspection Finds Problems.” Software Magazine: July 2005.

About the Authors

Dr. Jeffrey Carver is an Assistant Professor in the Computer Science and Engineering Department at Mississippi State University. His research interests include software process improvement, software quality, software inspections, and software engineering for high performance computers. Carver’s research has been funded by the U.S. Army Corps of Engineers, the U.S. Air Force, and the National Science Foundation. He received his doctorate from the University of Maryland in 2003.

Department of Computer Science and Engineering
300 Butler Hall
Box 9637
Mississippi State University, MS 39762
Phone: (662) 325-0004
Fax: (662) 325-8997
E-mail:carver@cse.msstate.edu

Dr. Forrest Shull is a senior scientist at the Fraunhofer Center for Experimental Software Engineering, Maryland (FC-MD). At FC-MD he is project manager and member of the technical staff for projects with clients that have included Fujitsu, Motorola, NASA, and the U.S. Department of Defense. He is responsible for research projects in the areas of software defect reduction and software practice evaluation. He received his doctorate degree from the University of Maryland, College Park.

Fraunhofer Center for Experimental Software Engineering Maryland
University of Maryland
4321 Hartwick RD STE 500
College Park, MD 20742-3290
Phone: (301) 403-8970
Fax: (301) 403-8976
E-mail: fshull@fc-md.umd.edu

Dr. Ioana Rus is a scientist at the FC-MD where she serves as a technical area lead for safety and security. Rus has represented FC-MD as a member of the Software Assurance Processes and Practices Working Group and as a reviewer for the DHS Software Assurance Common Body of Knowledge. She received her doctorate from Arizona State University.

Fraunhofer Center for Experimental Software Engineering Maryland
University of Maryland
4321 Hartwick RD STE 500
College Park, MD 20742-3290
Phone: (301) 403-8971
Fax: (301) 403-8976
E-mail: irus@computer.org

CrossTalk
The Journal of Defense Software Engineering

Article Index Volume 18

TOPIC	ARTICLE TITLE	AUTHOR(S)	ISSUE	PAGE
Acquisition	Cooperative Appraisals for Capability and Risk Evaluation	Diane A. Glaser, Michael D. Barnett	4	14
Ada 2005	Ada 2005	Richard L. Conn	8	4
	The Ada 2005 Language Design Process	S. Tucker Taft	8	20
	Ada 2005: A Language for High-Integrity Applications	Dr. Benjamin M. Brosgol	8	8
	Ada 2005 on .NET and Mobile and Embedded Devices	Dr. Martin C. Carlisle	8	16
	Intuitive Multitasking in Ada 2005	Dr. Bo I. Sanden	8	12
Agile	Army Simulation Program Balances Agile and Traditional Methods With Success	LTC John Surdu, Ph.D., Douglas J. Parsons	4	4
	Lessons Learned Using Agile Methods on Large Defense Contracts	Paul E. McMahon	5	25
Architectures	DoDAF-Based Information Assurance Architectures	Dr. John A. Hamilton, Jr.	2	4
Change Management	A Manager's Guide to Supporting Organizational Change: 10 Lessons Learned	Esther Derby	1	17
	Managing Cultural Changes in Your Organization	Dr. Kenneth D. Shere	4	9
	Transforming Cultures: A New Approach to Assessing and Improving Technical Programs	Hile Rutledge, Jennifer Tucker	1	20
Communication	Building Successful Software Development Teams Using TSP and Effective Communication Networks	Dr. William R. Nichols	1	12
	E-Mail Etiquette	COL Kenneth L. Alford, Ph.D.	1	16
	Safety Check	Steven M. Smith	1	10
	The Project Charter – Blueprint for Success	Chuck McKeever	1	6
	What We've Got Here Is Failure to Communicate	Alan C. Jost	6	10
Design	A Gentle Introduction to Object-Oriented Software Principles	Maj. Christopher Bohn, Ph.D., John Reisner	10	24
Estimation	Project Estimation With Use Case Points	Roy K. Clemmons	2	18
	Software Cost Estimating: A Cyclical Conundrum	Ellen Walker	7	Online
	Software Estimating Models: Three Viewpoints	Dr. Randall W. Jensen, Lawrence H. Putnam, Sr., William Roetzheim	2	23
Language	Maintaining Sanity in a Multilanguage World	Val C. Kartchner	8	22
Management	All We Need to Know About Sotware Project Management, We Can Learn From Watching Star Trek	David R. Webb	10	28
	Are Management Basics Affected When Using Agile Methods?	Paul E. McMahon	11	4
	Becoming A Great Manager: Five Pragmatic Practices	Esther Derby	11	9
	Leadership, The Final Frontier: Lessons Learned From the Captains of Star Tre	Paul Kimmerly, David R. Web	10	13
	Uncommon Techniques for Growing Effective Technical Managers	Paul E. McMahon	11	Online
Miscellaneous	A Brighter Future From Gallium Nitride Nanowires	Dr. Kris A. Bertness, Dr. Norman A. Sanford, Dr. Albert V. Davydov	10	9
	The *Science* in Science Fiction's Artificial Men	Dr. Dawn MacIsaac, Dr. Kevin B. Englehart	10	4
	Ten Lessons Learned: Data Warehouse Development Project, California Department of Fish and Game	Crilly Butler, Jr.	10	16
	What Science Fiction Authors Got Wrong and Why We're Better Off For It	Maj. Christopher Bohn, Ph.D.	10	Online
	When Did Six Sigma Stop Being a Statistical Measure?	Joe Schofield	4	28
Net-Centricity	Development of a Ground Vehicle Maneuver Ontology to Support the Common Operational Picture	Dr. Paul W. Richmond, Curtis L. Blais, Dr. Niki C. Goerger	7	26
	From the DoD CIO: The Net-Centric Information Enterprise	John G. Grimes	7	4
	Information Sharing Is a Strategic Imperative	Gen. James E. Cartwright	7	7
	The New Java Security Architecture	Idongesit, Mkpon-Ruffin, Dr. John A. Hamilton, Martin C. Carlisle	7	Online
	Overview of the Department of Defense Net-Centric Data Strategy	Anthony J. Simon	7	21

Continued on Next Page

From Previous Page

TOPIC	ARTICLE TITLE	AUTHOR(S)	ISSUE	PAGE
Net-Centricity (continued)	Secure From the Start: Designing and Implementing an Assured National Security Enterprise	LTG Keith B. Alexander	7	10
	Service-Oriented Architectures in Net-Centric Operations	Lt. Gen. Charles E. Croom, Jr.	7	13
	The Team: Creating the Enabling Capability to Conduct Net-Centric Operations	Lt. Gen. Robert M. Shea	7	18
	Transformational Communications Systems for DoD Net-Centric Operations	Dr. Troy Meink	7	23
Process Documentation	Defining Short and Usable Processes	Timonthy G. Olson	6	24
Process Improvement	Commandments for a Productive Development Environment	Dr. Randall Jensen, Les Dupaix	1	25
	Should Your Projects' Leaders Be on Springer?	Paul Kimmerly	6	29
Project Management	Earned Schedule: An Emerging Enhancement to Earned Value Management	Walt Lipke, Kym Henderson	11	26
	A Governance Model for Incremental, Concurrent, or Agile Projects	Dr. Alistair Cockburn	2	13
	Practical Performance-Based Earned Value	Paul J. Solomon	5	20
	Social and Technical Reasons for Software Project Failures	Capers Jones	6	4
	Start With "Simple" Earned Value on All Your Projects	Quentin W. Fleming, Joel M. Koppelman	6	16
	Statistical Methods Applied to EVM: The Next Frontier	Walt Lipke	6	20
PSP/TSP	Designing in UML With the Team Software Process	David R. Webb, Ilya Lipkin, Evgeniy Samurin-Shraer	3	14
	Factors Affecting Personal Software Quality	Dr. Mark C. Paulk	3	9
	Maturing the PSP: Developing A Body of Knowledge and Professional Certification for PSP-Trained Software Developers	Dr. Marsha Pomeroy-Huff	3	19
	Using TSP With a Multi-Disciplined Project Managment System	Timothy A. Chick	3	4
Requirements	Experiences in Eliciting Security Requirements	Dr. Nancy R. Mead	12	14
	Interpreting Requirements in a He Said/She Said World	Deb Jacobs	12	9
	Requirements as Enablers for Software Assurance	Dr. Seok-Won Lee, Robin A. Gandhi	12	20
	Twelve Requirements Basics for Project Success	Dr. Ralph R. Young	12	4
Quality	Exposing Software Field Failures	Michael F. Siok, Clinton J. Whittaker, Dr. Jeff Tian	11	15
	Finding and Fixing Problems Early: A Perspective-Based Approach to Requirements and Design Inspections	Dr. Jeffrey Carver, Dr. Forrest Schull, Dr. Ioana Rus	12	26
	Implementing Phase Containment Effectiveness Metrics at Motorola	Ross Seider	11	12
	Integrated Quality Assurance for Evolutionary, Multi-Platform Software Development	Dr. Robert B.K. Dewar	11	Online
	Software Recapitalization Economics	David Lechner	11	21
	Using Line of Balance to Track the Progress of Fixing Trouble Spots	Eduardo Miranda	4	23
Reuse	How to Relate Quality and Reuse in Evolving Systems	Dr. Ronald J. Leach	4	27
Software Assurance	Adapting Legacy Systems for DO-178B Certification	Paul R. Hicks	8	27
	Applying RAMS Principles to the Devlopment of a Safety-Critical Java Specification	Dr. Kelvin Nilsen	2	8
	Availability, Reliability, and Survivability: An Introduction and Some Contractual Implications	Dr. Jack Murphy, Dr. Thomas Ward Morgan	3	26
	Increasing the Likelihood of Success of a Software Assurance Program	Steven F. Mattern	9	26
	The Qualification of Software Development Tools From the DO-178B Certification Perspective	Dr. Andrew J. Kornecki, Dr. Janusz Zalewski	4	19
	When Computers Fly, It Has to Be Right: Using SPARK for Flight Control of Small Unmanned Aerial Vehicles	Lt. Col. Ricky E. Sward, Ph.D., Lt. Col. Mark J. Gerken, 2nd Lt. Dan Casey	9	10
Software Security	Assessing Information Security Risks in the Software Development Life Cycle	Douglas A. Ashbaugh	9	21
	Building Multilevel Secure Web Services-Based Components for the Global Information Grid	Dylan McNamee, CDR Scott Heller, Dave Huff	5	15
	Security in the Software Lifecycle	Joe Jarzombek, Karen Mercedes Goertze	9	4
Systems Engineering	A System View of Merging Software and Hardware	Mike McNair	10	21
Systems of Systems	21st Century Processes for Acquiring 21st Century Software-Intensive Systems of Systems	Dr. Barry Boehm, Jo Ann Lane	5	4
	Tackling the Cost Challenges of System of Systems	Arlene F. Minkiewicz	5	10
Testing	Application and Evaluation of Built-In Test (BIT) Techniques in Building Safe Systems	James A. Butler	9	15
	Understanding the Logic of System Testing	Dr. Yuri Chernak	3	22
Training	Hard Skills Simulations: Tackling Defense Training Challenges Through Interactive 3-D Solutions	Josie Simpson	2	Online
	Knowledge: The Core Problem of Project Failure	Timothy K. Perkins	6	13

Net-Centric Warfare Changes Poetry

A cruel English teacher once made me put together a poetry anthology, which is a collection of my favorite poems. Now, who would do that to a high school kid? Anyhow, after all these many years, I still have it, and came across it the other day. Many of our great songs and poems have military origins, or military subjects.

The Charge of the Light Brigade is in my anthology, so I can "Honor the charge they made."

That poem was based on a real battle that should never have happened. The order was sent for them to retreat, but, as the poem says, *Someone had blundered*. They received the word to charge, and they did. This wouldn't have happened in a net-centric world and we wouldn't have great misquotes like "Ours is not to reason why, ours is but to do and die." I heard that many times when I was in the military, and I hear it occasionally now. The poem actually reads *Their's not to reason why, Their's but to do and die*.

I think I remember reading that this verse is the most often misquoted one in history, but I don't know how one would determine that.

My forced anthology has another, less well known, humorous poem, that deals with communication on the battlefield, and the human nature of warfare. It is called "Pershing at the Front," and was written by the humorous poet Arthur Guiterman. World War I was fought in *the murk and the powder stench*, and the *wet and the muck as well*. Will net-centric warfare do away with this colorful environment? Or will it bring a shared awareness between those with boots on the ground and those who are two or three thousand miles away remotely applying mass while sitting in an air conditioned environment? As well as being in my poetry anthology, this wonderful poem can be found at <http://holyjoe.org/poetry/guiter8.htm>. The ending highlights another aspect of war that just can't be replaced by modern technology.

Now, instead of all the world wondering about the wild charge, all the world is wondering about net-centric warfare, and what it will do to the nature of war. I'm wondering about what it will do to the nature of song, poetry, and our society.

Now we are going to have to write poems about being self-synchronized and having shared awareness. And how about this *shared awareness* stuff? If we would have had shared awareness in the 1800s, literature might be lacking other famous prose.

What about let's *remember the Alamo*, and the great ballad that came from that battle? What would have some shared awareness done for that band of heroes? Did Bowie and Travis know that reinforcements were not going to come? Would it have mattered?

I fear that this change in warfare will have a negative impact of the world of poetry and song. We are going to have fewer poems and songs about love and brave deeds. Do we really want to pursue this technology?

— Dennis Ludwig
dennis.ludwig@wpafb.af.mil

Monthly Columns:

Issue		Column Title	Author
Issue 1: January Communication	Sponsor: Publisher: BackTalk:	Are Your Communication Skills Advancing? Communications: Continuous Improvement Required Who's on Project Management?	Bob Zwitch Tracy L. Stauder Gary A. Petersen
Issue 2: February A New Twist on Today's Technology	Sponsor: Publisher: BackTalk:	What Is Up and Coming Improving a Little at a Time Tech-Neologism	Kevin Stamey Elizabeth Starrett Gary A. Petersen
Issue 3: March PSP/TSP	Sponsor: BackTalk:	Software Product Development: Transforming Art to Science Why Isn't There an "I" in *Team*?	Terrence Clark Dr. David A. Cook
Issue 4: April Alternate Mixes for CMMI	Sponsor: BackTalk:	Transitioning to a New Model? First Consider Your Organizational Culture Win the Battle, Lose the War	Randy B. Hill Dr. David A. Cook
Issue 5: May Transforming: Business, Security, Warfighting	Sponsor: Publisher: BackTalk:	Transformation: A Continuous Process Education Is Key for Successful Transformation Transform This	Bob Zwitch Elizabeth Starrett Gary A. Petersen
Issue 6: June Why Projects Fail	Sponsor: BackTalk:	Why Do Projects Fail? When Failure *IS* an Option ...	Kevin Stamey Dr. David A. Cook
Issue 7: July Net-Centricity	Sponsor: BackTalk:	The Future Battlespace and the Power of Immediate Decision Making e-Dorado: The Lost Centric City of Information	Terrence Clark Gary A. Petersen
Issue 8: August Ada 2005	Sponsor: Publisher: BackTalk:	CrossTalk Co-Sponsors Invite Additional Government Organizations on Board Ada Continues to Evolve Ada: The Maginot Line of Languages	Randy B. Hill Elizabeth Starrett Dr. David A. Cook
Issue 9: September Software Assurance	Sponsor: BackTalk:	Software Assurance: Highlighting Changes Within Our Software Community of Practice Ready, Fire, Aim!	Joe Jarzombek Gary A. Petersen
Issue 10: October Star Wars to Star Trek	Sponsor: Publisher BackTalk:	The Vision of What Can Be Star Wars, Star Trek, and CrossTalk Science Fiction, Science Fact	Diane E. Suchan Elizabeth Starrett Dr. David A. Cook
Issue 11: November Management Basics	Sponsor: Publisher: BackTalk:	A Willingness to Keep Learning Basic Articles Deja Review	Kevin Stamey Elizabeth Starrett Gary A. Petersen
Issue 12: December Requirements Engineering	Sponsor: BackTalk:	Requirements Management Is Required Net-Centric Warfare Changes Poetry	Randy B. Hill Dennis Ludwig

Building Solutions for the Systems of the Past, Present, and Future!

CMMI Level 5

AS9100 ISO 9001

Please contact us today

Ogden Air Logistics Center
309th Software Maintenance Group
(formerly MAS Software Maintenance Division)
Hill Air Force Base, Utah 84056

Commercial: (801) 777-2615, DSN 777-2615
E-mail: ooalc.masinfo@hill.af.mil
or visit our website: www.mas.hill.af.mil

CrossTalk is co-sponsored by the following organizations:

Homeland Security

NAVAIR

CrossTalk / 517 SMXS/MXDEA
6022 Fir AVE
BLDG 1238
Hill AFB, UT 84056-5820

PRSRT STD
U.S. POSTAGE PAID
Albuquerque, NM
Permit 737